MASTER'S THESIS | LUND UNIVERSITY 2017

Constructing a Multilingual Relation Extraction System Using Neural Networks

Erik Gärtner, Axel Larsson

Department of Computer Science
Faculty of Engineering LTH

ISSN 1650-2884
LU-CS-EX 2017-15

Constructing a Multilingual Relation Extraction System Using Neural Networks

Erik Gärtner
erik.gartner.770@student.lu.se

Axel Larsson
axel.larsson.534@student.lu.se

September 1, 2017

Master's thesis work carried out at Sony Mobile Communications AB.

Supervisors: Pierre Nugues, pierre.nugues@cs.lth.se
Håkan Jonsson, hakan1.jonsson@sonymobile.com

Examiner: Jacek Malec, jacek.malec@cs.lth.se

Abstract

A large amount of information is available on the internet in the form of natural language in written form. In order to make the information useful for computer systems, such as knowledge graphs and question-answering systems, the information must first be structured. In this thesis, we describe a multilingual system that in a semi-unsupervised manner can learn to extract certain types of stated facts. To achieve this, the system uses natural language processing techniques with neural networks and word embeddings.

Our evaluation shows promising results; it achieves good precision on selected relations and further improvements are most likely possible in the future.

Keywords: natural language processing, information extraction, fact extraction, knowledge base, word embeddings, neural networks

Acknowledgements

We would first like to thank our amazing supervisor Professor Pierre Nugues. His deep knowledge within the field of natural language processing and his helpfulness was invaluable. More so his encouragements that kept our spirits high when we encountered problems.

Next we would like to thank PhD student Marcus Klang who provided his incredible technical expertise in a wide range of areas from Apache Spark to natural language processing and Java garbage collection. Without his Docforia document model library and all the data he provided us this thesis would have been an herculean task to complete in a semester. Finally Marcus was always just a knock on the door away when the cluster crashed or we needed help.

Our supervisor at Sony Mobile, PhD student Håkan Jonsson, also deserves our gratitude. Without him this thesis simply would not have existed. It was his idea that set this entire thesis in motion. Throughout the entire process he has been very helpful and did his utmost to provide us with the tools and resources we needed.

We would also like to extend a thank you to Professor Jacek Malec for taking on the role as our examiner and for igniting our interest in artificial intelligence in his course: *Applied Artificial Intelligence*.

No acknowledgements would be complete without thanking our parents. Anders and Cecilia Gärtner and Staffan and Carina Larsson for their support during our entire lives and for encouraging our interest in computer science. Axel would also like to thank his siblings, Anna and Ulf, for their presence in his life that makes it that much more interesting and enjoyable. Finally we would like to thanks Anders and Ceclia for proofreading this report.

Contents

Chapter 1
Motivation

Whoever is careless with truth in small matters cannot be trusted in important affairs.

Albert Einstein (Holton, 2016)

During the 2016 U.S. presidential election the world witnessed an unprecedented amount of fake news. The aim of these news seemed to be to generate ad revenue (Kirby, 2016) as well as to disrupt democratic processes (Timberg, 2016). Before the rise of digital media journalists could fact-check articles as they were published but today the sheer amount of blog posts and articles make this almost impossible.

In order to combat fake news, we propose creating a large-scale and trustworthy database of facts. This knowledge base could then later be used to check facts in social media, articles and blog posts. Our belief is that detecting fake and misleading news needs to based on verifying stated facts.

Chapter 2
Introduction

In this chapter we will cover the background required to understand this thesis. Since the fields of natural language processing, computer science and machine learning span decades we only briefly cover the parts most relevant to the readers' understanding. Furthermore the chapter outlines the problem statement at the heart of this thesis.

2.1 Background

This section provides the user with a short introduction to the areas of knowledge bases, natural language processing, and machine learning.

2.1.1 Knowledge bases

A **knowledge base** refers to a database containing information, or knowledge, in various formats. In this thesis we will use it to refer to databases of structured information.

One prominent knowledge base was **Freebase** (Google, 2017) which was acquired by Google and subsequently shut down. It was succeeded by the open **Wikidata** (Wikidata, 2017a). Both contain data in the form of structured relations triple in **RDF** format (see section 2.1.3).

2.1.2 Information Extraction

In the computer science field of **natural language processing** creating structured data from unstructured data is usually called **information extraction**.

Natural language processing deals with constructing programs and algorithms that can understand, synthesise and model human language. There are multiple approaches but we will focus on the statistical approach. By feeding large amounts of data to machine learning algorithms, we can create statistical models that for example can be used to extract information from sentences.

Constructing knowledge bases

One common method for constructing a knowledge base is to gather information from **unstructured** text sources. They are called unstructured because the data does not conform to a predefined data schema. An example of unstructured text would be books and most web pages. Structured text would be the semantic web or data formats such as RDF.

2.1.3 Resource Description Framework

In the W3C format **resource description framework**, or RDF, information is expressed in the form of **triples** that describe relations, called **predicates**, between two **entities** (W3C Foundation, 2014). For example the sentence "Barack Obama is married to Michelle Obama" would be represented as (Barack Obama, married to, Michelle Obama), where "Barack Obama" is the subject, "married to" is the predicate, and "Michelle Obama" is the object. In RDF entities and predicates have unique and language independent identifiers which makes the information unambiguous and better structured for analysis.

Querying the data for example becomes trivial. If we want to find all information about Barack Obama we simply retrieve all the triples containing Barack Obama as either the subject or object. Furthermore if we want to study all the marriages we retrieve all the triples containing the "married to" predicate. In fact the triples can be represented as a knowledge graph were the entities are nodes and the predicates are arcs.

As such it would be very beneficial to convert unstructured text into structured text. One method is to **extract** RDF triples from the information in an unstructured text. If this can be accomplished then the vast amount of texts in books and on the Internet could be stored in a knowledge base.

2.2 Previous work

In *Distant Supervision for Relation Extraction Without Labeled Data* (Mintz et al., 2009) introduce a novel approach for creating relation extraction models. By using Freebase to create labelled examples from an unstructured database (such as Wikipedia) they are able to generate large amounts of training data to train high precision models for relation extraction. Since then multiple papers using and refining Distant Supervision Learning has been published.

Zeng et al. (2015) experimented with using convolutional neural networks as well as new features to handle the inherent problem with distant supervision extracting false positives during training data generation.

Quirk and Poon (2016) addressed the limitation in distant supervision where learning requires that the relations must exist in the same sentence. In their paper *Distant Supervision for Relation Extraction beyond the Sentence Boundary* they explore the possibility of extracting relations over sentences boundaries with promising results. Like many other papers they apply their model to the biomedical domain.

Dong et al. (2014a) further the work of Mintz et al. in their paper *Knowledge Vault: A Web-scale Approach to Probabilistic Knowledge Fusion* by creating thousands of these relation extraction models and then using them to create a knowledge graph. The paper puts heavy emphasis on resolving inconsistencies and handling incorrect extraction by the model (called **knowledge fusion** in the paper).

2.3 Contributions

Despite many papers building on distant supervision we address the challenge of building a multilingual system which is relatively unexplored. Our system in itself is not language-dependent. As long as the underlying tools used to tokenise and parse the corpus support the language our system supports the language.

The original paper by Mintz et al. (2009) uses a simpler approach consisting only of logistic regression models. Our model uses a neural network for classification of the sentence in addition to a logistic regression model for filtration.

Furthermore our model uses Word2vec word embeddings to represent the lexical features which adds additional information about the meaning of the lexical features. This has the benefit of turning similar word almost into word classes, since for example all the weekdays will be close to each other in the vector space.

2.4 Problem

When working with information in computer science it is preferable to work with structured data because it makes the information less ambiguous and makes information extraction easier. Unfortunately the vast majority of information on the internet is in the form of unstructured text, for example in encyclopedias and news articles.

Manually adding all facts contained in unstructured text into a structured format would be a daunting and time consuming task. Therefore it would be of tremendous benefit if we could leverage computers to automatically extract facts from the unstructured text into a structured database.

The main problems faced when extracting facts from unstructured text are:

- **Identifying the facts contained in a sentence.** For examples what facts are contained in the sentence *Barack Obama was born in Honolulu*.
- **Uniquely identifying the entities in a sentence.** E.g. *Barack Obama* and *Honolulu*.
- **Resolving inconsistencies between different sources and extraction errors.**
- **Merging facts from multiple languages.**
- **Handling errors in the source texts.**

2.4.1 Scope of This Work

In this report we are limiting our scope to creating multilingual extraction models with high precision for Swedish and English. We will not focus on resolving inconsistencies based on source error or develop methods for merging facts from the different languages.

2.4.2 Work Division

Developing of the main system was shared between both authors. The fact-checker module was written by Axel and the Chrome plugin was written by Erik. Furthermore Erik was responsible for executing and managing the model training. Both authors performed the manual evaluation.

In the report Erik wrote most of chapter two as well as the machine learning theory in chapter three. Axel wrote the rest in chapter three and as well as most of chapter four. The rest of the report was written jointly.

Chapter 3

Approach

In this chapter we will outline the approach we took to solving the problem defined in the previous chapter.

3.1 Theory

This section will introduce the theoretical background needed to understand our methodology and implementation.

3.1.1 Machine Learning

There exists a myriad of different algorithms and approaches to **machine learning**. What they all have in common is that they all tune parameters of a function to minimise a given objective function. That is, they try to predict the behaviour of a function by minimising some error between the predictions and the actual values.

The process of tuning the parameters and fitting the model to data is called **training**.

The are two types of predictions, **classification** and **regression** (Bishop, 2009, ch. 1). Classification deals with assigning data points into discrete groups. For example labelling an example as "good" or "bad". Regression on the other hand tries to predict continuous values such as predicting the stock value for the next day.

Machine learning algorithms are divided into **supervised** and **unsupervised** learning. During supervised learning the algorithm is shown labelled examples from which it is supposed to learn the parameters to best predict the label given an input. Labelled means that it is shown both the input values as well as the correct output value. Unsupervised learning on the other hand does not provide any labels. These methods strive to cluster similar data points together into groups (Bishop, 2009, ch. 1).

Another way to express machine learning is to describe it as the process of creating a model by observing data in order to capture underlying patterns. As such the quality as well as the quantity of data have a great impact on the performance of the final model. A common expression is "garbage in garbage out", which expresses that if the training data is of low quality then the model will be of equally low quality.

3.1.2 Features

In machine learning **features** are the different inputs of the model. In Table 3.1 each row is a **data point** (also called **example**). "Headache" and "Temp" are features and the label is "Has Disease A". From this data the algorithm is supposed to create a model that can predict whether a patient has Disease A given whether or not they have a headache and their body temperature.

Headache	Temp (C)	Has Disease A
Yes	40,5	Yes
No	41,0	No
No	37,1	No
Yes	37,2	No

Table 3.1: A toy example demonstrating the features "headache", "temp", and how they relate to the label "has Disease A".

The "Headache" feature is a **categorical feature**; it is limited to a finite set of possible values, while "Temp" is a **numerical feature**. Likewise *Has Disease A* is also a categorical variable implying that this is a classification problem, not a regression problem.

Selecting features is one of the most important aspects of machine learning since it directly affects what information is available to the algorithm. It is very likely that a complex disease can't accurately be diagnosed with only two simple features.

3.1.3 NLP Features

In natural language processing there exists a few very common types of features.

Words are the most simple types of features.

Part-of-speech features are the part-of-speech tags of the words, for example "noun", "verb" and so forth (Nugues, 2006, p. 113).

Entity types are features describing the type of an "entity word". For example the words "Barack Obama" has the type "PERSON" and "Lund, Sweden" has the type "LOCATION".

Dependency Parse Tree

Another feature that is useful in NLP applications is a composite feature; the dependency parse. It is a set of words that is linked by their directional dependencies. Each word in the sentence is tagged with a part-of-speech tag (e.g. verb, noun) describing its syntactic function. The syntactic relationship between the words is then represented by the dependency parse tree. For example, the sentence "Barack married Michelle" has the dependency parse tree that can be seen in Figure 3.1.

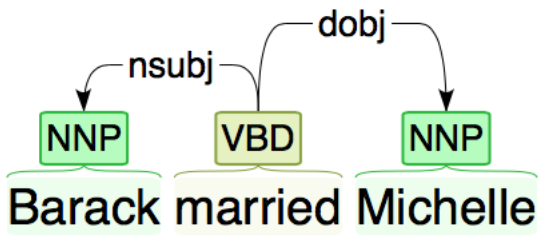


Figure 3.1: An example dependency parse tree.

As a feature we extract the path from the dependency parse tree that goes from the first entity to the second. This is achieved through a depth-first-search algorithm that searches the tree from the first entity, targeting the second entity. At each step, the word, its dependency and the direction are recorded. From the example in Figure 3.1, the dependency path would be "true/nsubj/married, false/dobj/Michelle", where the Boolean value indicates the direction.

3.1.4 One-Hot Encoding

Machine learning algorithms are mathematical models that often require numerical representations of all features, including the categorical features. One very common and simple method to encode words as numerical vectors is called **One-Hot Encoding**.

All possible values for the categorical feature are assigned an index from 1 to N, where N is the number of possible values. Then each value is assigned a corresponding vector of length N where all elements are 0 except the i:th element which is 1 ("hot"), where i is the index of the value. See Table 3.2 for an example.

Value	**Value number**	**One-Hot encoded vector**
Hello	1	[1, 0]
World	2	[0, 1]

Table 3.2: A toy example demonstrating One-Hot Encoding for a word feature containing only two different possible values.

3.1.5 Coreferences

A **coreference** can occur in a text when there are two or more statements in a text referring to the same entity (Nugues, 2006). The mentions of the same entity are said to be **co-referential** and one of the mentions is the **antecedent**, the full form of the entity, while the others are called **anaphora**, usually in the form of pronouns.

One can follow the anaphora back to the antecedent and reach the named entity, resolving the mentions to a single named entity. Consider the example document in Figure 3.2. Here, *Barack Obama* is mentioned twice, first in the full form; the antecedent, and then in an abbreviated form, “He” – the anaphor.

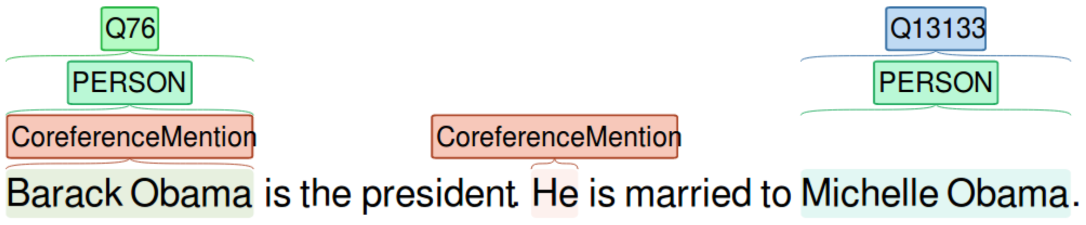

Q76
PERSON
CoreferenceMention
CoreferenceMention
Q13133
PERSON
Barack Obama is the president He is married to Michelle Obama.

Figure 3.2: An example demonstrating coreferences.

3.1.6 Word2vec Word Embeddings

One-Hot Encoding is a simple method but the vectors produced are arbitrarily assigned and do not contain any meaning. They also quickly grow to very large sizes for language tasks. For example encoding the 100,000 most common English words turns each feature into a vector of dimension 100,000.

Word2vec is a feature representing technique where the vector representation is learnt from a large corpus using a neural network (Mikolov et al., 2013). The vectors are generally small, about 300-500 dimensions. The most significant feature of Word2vec is that the word embeddings are close to each other in the vector space. That is they are clustered in the vector space by their semantic meaning. This provides our machine learning model with useful information about how the values of the feature relate to each other. For example if we look at the word “Sweden”, its closest neighbours are likely other Scandinavian countries such as “Norway” or “Denmark” and a completely unrelated word such as “screwdriver” is far away.

Furthermore the vectors have semantic meaning. In a well trained network for example this arithmetic will be valid:

$$vector(''King'') - vector(''Man'') + vector(''Woman'') \approx Vector(''Queen'')$$

3.1.7 Evaluation

In machine learning, evaluation of the model is of great importance, since it is how the result of the training and the performance of the model is measured.

Because the training process involves fitting the model to the training dataset it is to be expected that the model improve with each training iteration. This does not however imply that the model improves for predicting previously unseen data points.

The behaviour of improving on predicting labels for data points in the training dataset but not for similar data points not previously seen is called **overfitting**.

To evaluate whether or not the model actually learnt anything useful and not just overfitted there exist two common methods.

The first method is to withhold parts of the training dataset from the model during training, this subset is called the **validation dataset**. After the model is trained it is applied to the validation dataset to predict their labels. The predictions are then compared to the labels. This method is called **cross-validation** (Russell and Norvig, 2009, ch. 18.4).

The second method is to have a completely different dataset called **test dataset**. This dataset should not be a subset of the training dataset, making the evaluation of this dataset even less biased.

Metrics

There exist many different metrics for calculating the performance of the model. Since our thesis deals with an information extraction problem we will cover metrics applicable to that problem and model type (Nugues, 2006, ch. 9.9).

True Positive is the number of positive examples labelled positive. In the example from Table 3.1 it would correspond to the number of patients that have “Disease A” and that are also diagnosed with having it.

False Positive is the number of negative examples labelled negative. This corresponds to the patients not having “Disease A” diagnosed with having it.

False Negative is the number of positive examples labelled negative. That is the number of patients having “Disease A” diagnosed with not having it.

True Negative is the number of negative examples labelled negative. That is the number of patients not having “Disease A” diagnosed as not having it.

Recall is the percentage of positive examples labelled as positive.

Precision is the percentage of positive labelled examples that are actually correctly classified.

F-measure / F1-score is the harmonic mean between recall and precision. It is a composite metric that does not favour any of the two metrics. The reasoning behind the F1-score is that getting either 100% precision or 100% recall is easy. To get 100% precision simply label no examples as positive and you will never be wrong. Similarly to get 100% recall just classify all examples as positive. The F1-score requires both recall and precision to be high to get a high F1-score.

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} \tag{3.1}$$

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} \tag{3.2}$$

$$\text{F1-score} = \frac{\text{Recall} \cdot \text{Precision}}{\text{Recall} + \text{Precision}} \tag{3.3}$$

Figure 3.3 demonstrates the metrics using the same patient example as in Table 3.1.

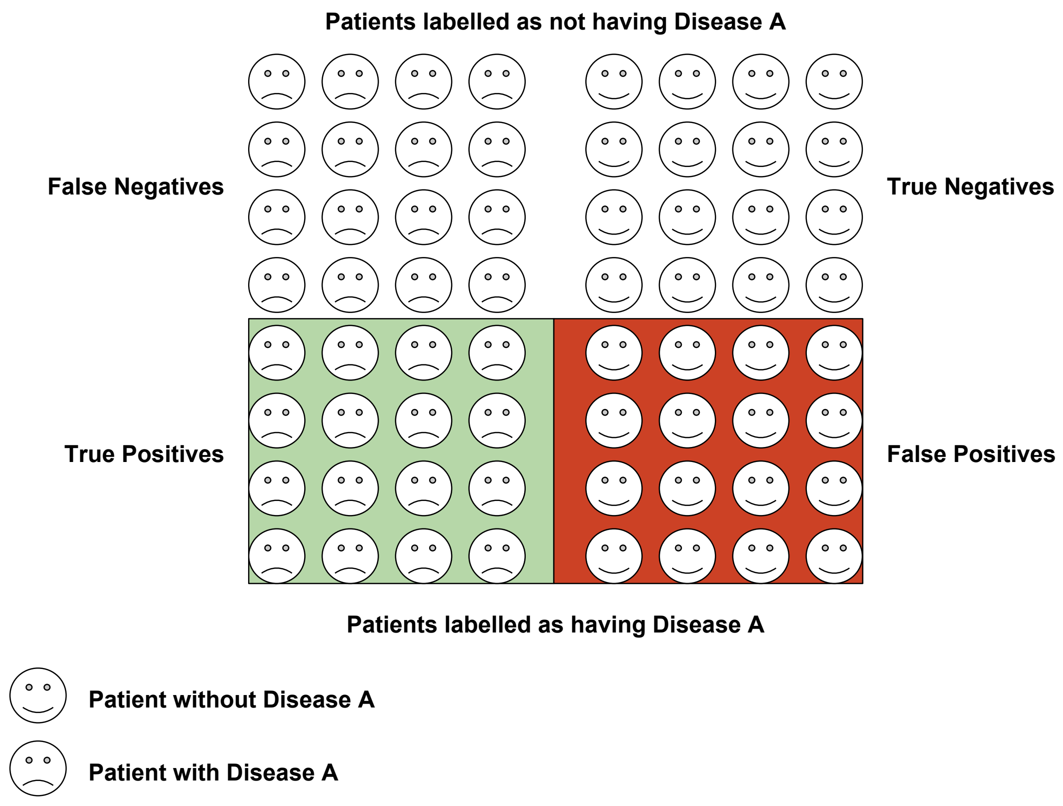

Patients labelled as not having Disease A
False Negatives
True Negatives
True Positives
False Positives
Patients labelled as having Disease A
Patient without Disease A
Patient with Disease A

Figure 3.3: The figure shows a visual demonstration of the different evaluation metrics.

3.1.8 Logistic Regression

Logistic regression is a supervised classification algorithm using the logistic sigmoid function (Bishop, 2009). Logistic regression is commonly used when there is a need to model the probabilities of k classes via a linear function that sums to one and the output for each class is within [0, 1] (Hastie et al., 2013).

These outputs can then be used as probabilities for each class, thus training the logistic regression model to predict the probability of each class.

3.1.9 Neural Networks

Neural networks are nonlinear statistical models (Hastie et al., 2013, ch. 11.3). They were originally an attempt to create a model similar to how the neurons in the brain interact (Bishop, 2009, ch. 5). Neural networks are generally supervised models that can be used for either classification or regression. Because the model is nonlinear it can learn and approximate nonlinear functions which in theory makes the model a powerful tool.

In Figure 3.4 we see the schematic of a neural network. The most simple version is the **feed-forward network** where the network consists of one input layer, one or more hidden layers, and one output layer. Each “neuron” (the nodes in the schematic) in each layer are simple mathematical functions applied to the output of every neuron in the previous layer.

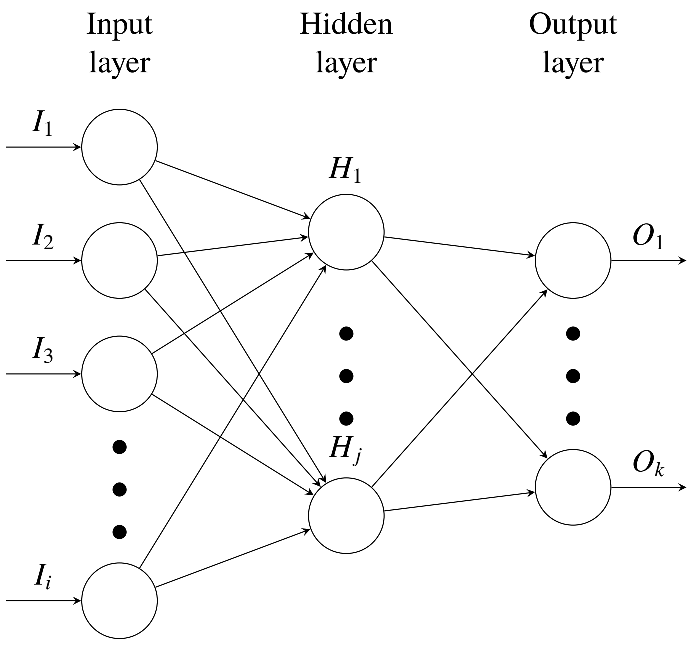


Figure 3.4: Schematic drawing of a feed-forward neural network with a single hidden layer.

The mathematical function applied to the outputs of the previous layer are called **activation functions**.

When designing a neural network several hyperparameters have to be taken into account. For example the number of neurons in each layer, the number of layers and the selection of activation functions for each layer.

Deep networks are those with many hidden layers.

3.1.10 Cluster Computing

We process a large amount of information in our pipeline; the whole of the English Wikipedia is used both to train our extractor on and to perform extractions over. Thus, due to the sheer volume of data that flows through our program, the pipeline needs to be run on a cluster computing platform to achieve acceptable performance.

MapReduce

One popular approach to processing big data in parallel on a cluster is called the MapReduce framework (Dean and Ghemawat, 2008). It was originally developed as a proprietary Google technology but now the term is used in a broader sense for frameworks incorporating that programming paradigm. One popular open source MapReduce framework is Hadoop MapReduce (The Apache Software Foundation, 2017a). Hadoop is a project for distributed computing consisting of several modules:

Hadoop HDFS is a distributed file system.

Hadoop YARN is a framework for job scheduling and cluster management resources.

Hadoop MapReduce is a MapReduce framework for parallel processing of large datasets.

MapReduce is an algorithm for processing and generating big datasets in parallel on a distributed computing platform. The user, i.e. the developer, needs to specify two fundamental functions, the `map` and the `reduce` function. The map function performs filtering and sorting and the reduce function performs a summary operation. The promise of the MapReduce concept lies in utilising parallelism with the help of multithreading on the underlying cluster. The framework automatically distributes the data on the cluster's worker nodes, the "shuffle step"; such that tasks can be done in parallel. The data is distributed based on the key as defined by the map operation. Thus, all data belonging to one key will be on the same worker node. This incurs some communication overhead when data must be serialised and de-serialised across the network. As such, decreasing communication costs is paramount in order to gain good performance. In the reduce step the worker nodes process the data in parallel per key.

Apache Spark

Spark is a "fast and general processing engine compatible with Hadoop" (The Apache Software Foundation, 2017b). It was developed to improve upon MapReduce in a number of ways. MapReduce enforces a linear dataflow, where input data is read from disk, mapped, reduced and saved to disk. This slows things down and makes it difficult to implement iterative algorithms that are very important in machine learning applications. Apache Spark builds a so-called DAG, a directed acyclic graph of (mostly) in-memory tasks to model the dependencies between the map and reduce operations of the program. This provides better performance most of the time for certain applications such as iterative algorithms or interactive data mining.

The Spark core provides an in-memory distributed data structure called the "RDD" - **Resilient Distributed Dataset** (Zaharia et al., 2012) and an API to interact with it in Scala, Java, Python and R. An RDD is a lazy, immutable data structure that offers abstractions to interact with it in a natural way, especially in Scala. Consider the canonical word-count example; count the number of occurrences of each unique word in a set of documents. Writing this in Spark is straight-forward, see Listing 1.

```scala
// Read input from disk, hdfs in this example
val textFile: RDD[String] = sc.textFile("hdfs://...")

// sum ocurrences for each word
val counts: RDD[(String, Int)] = textFile
    .flatMap(line => line.split(" "))
    .map(word => (word, 1))
    .reduceByKey(_ + _)

// save the results to disk
counts.saveAsTextFile("hdfs://...")
```

Listing 1: Spark word count example

An RDD can contain any Scala, Java or Python data type, for example `RDD[String]` is an RDD of Strings. There are two types of operations that can be performed on an RDD, transformations and actions. Transformations are lazy and only describe what to do with the data, not when to do it. Actions are strict and force the computation of the transformations leading up to the resulting data structure. In the word-count example, the `flatMap`, `map` and `reduceByKey` operations are all transformations of the data and the only action is the last `saveAsTextFile` invocation, which actually forces the computation. Because RDD:s are lazy, Spark can optimise away certain operations since nothing is computed until the final result is required in the program.

There is also another core distributed collection of data called **Dataframe** which, like an RDD is lazy and immutable. Unlike an RDD however, it is untyped and the data is organised into named columns like in a relation database. A Dataframe can be queried with SQL-like queries and is suitable for certain types of organised data.

On top of the Spark core there are two machine learning libraries, `mllib` and `ml` where the former is built on top of RDD:s and the latter on Dataframe:s.

The Spark architecture is composed of a single master and multiple worker nodes in a cluster; a node is in this sense (usually) a physical machine in a cluster. A Spark application that is run on a cluster is run as independent sets of processes that are coordinated by the **driver program**, the main JVM process driving the execution, that is run on the master node. The master node is then connected to the worker nodes which provide **executors**, JVM processes that actually perform the computations and store the data for the application. The driver program sends the application code as a JAR file to the executors and then the executors are sent serialised **tasks** for them to run. The tasks are able to be de-serialised by the executors since they have a copy of the application JAR. Executors can run multiple tasks, both in parallel and sequentially.

Spark can be run on Hadoop YARN or it can be run in so-called standalone mode. Importantly Spark does not need to be run on a cluster but can also be run locally as long as the Spark dependency is met, which is straight-forward to provide locally since all it requires is the download of a single jar file. This makes it easier to develop and test locally, with a subset of the data, before submitting the application to the cluster.

3.2 Method

The first step in building a knowledge base is to extract facts that can be inserted into the knowledge base, to this end we build a **fact extractor**. The facts that we extract are structured as so-called **relation triples** in the form of (subject, predicate, object), for instance (Barack Obama, place of birth, U.S.A) is a relation triple stating the fact that Barack Obama was born in the U.S.A.

The extractor is built to recognise certain types of predefined relations and we use a subset of standardised relations defined by Wikidata (2017b). The relations we consider can be seen in Table 3.3.

Title	Wikidata ID	Subject	Object
spouse	P26	Person	Person
place of birth	P19	Person	Location
educated at	P69	Person	Organisation

Table 3.3: Supported relations and their Wikidata property numbers as well as the types of the participating entities.

Wikidata

Wikidata is a free and open multilingual knowledge base that can be edited by both humans and machines (Wikidata, 2017a). It acts as central storage for structured data from Wikipedia and other free sources. In Wikidata there is the concept of **items** that are used to represent the "things" in human knowledge, including concepts, topics, and objects. Items are unique and each item has an identifier prefixed with a "Q"; for instance "Q76" is the Q-number for Barack Obama and "Q34" is the Q-number for Sweden. Each item has its own Wikidata page that contain statements about the item such as "Barack Obama" is an "instance of human" and "is the spouse of" Michelle Obama. Furthermore, these statements about the entities are called properties and each type of property has a Wikidata identifier prefixed with a "P". For instance, the P-number for the *spouse* relation is P26.

Distant Supervision Learning

The approach we have taken to build the relation extractor is to use machine learning to build a model that learns to detect stated facts in text. To train this model we need to have a labelled training set. Ideally we would thus have a very large corpus of text where all the stated facts in the text are correctly labelled as relation triples. The model would then be fed this corpus and would learn from texts that state actual relations using them as positive examples and using texts that does not contain them as negative examples. For instance the sentence "Göran Persson is married to Anitra Steen" would be a positive example of a *married to* or *spouse* relation.

This would allow us to perform **supervised learning**; the task of learning from labelled data. Unfortunately there is no such large pre-labelled corpus but we would not like to resort to **unsupervised learning**; the task of learning from unlabelled data.

Instead we use **distant supervision learning** (Mintz et al., 2009). Distant supervision learning is a hybrid of supervised and unsupervised learning; it is a form of semi-supervised learning. The main idea is to compile a list of pairs of entities that participate as the object and subject in a relation and find all the occurrences of sentences that contain these pairs. For instance, we know from Wikidata statements that Barack Obama has Michelle Obama as spouse (and vice versa since it is a symmetrical relation). Thus when we find both Michelle and Barack as disambiguated named entities in a sentence, there is a high probability that that sentence actually states the *spouse* relation between the two entities and we can count that sentence as a positive example of a *spouse* relation. To be able to extract training examples we must annotate the input text with appropriate annotations such as part-of-speech tags and dependency parse trees. These annotations are used to extract features from the sentences.

The most important annotation however is the **named entity disambiguation**. A named entity is a real-world object such as a person, location, organisation; objects that can be denoted with a proper name. Disambiguation means that we must uniquely identify the named entity. For instance if we encounter a sentence mentioning Göran Persson we need to know whether it is the former prime minister of Sweden that is referred to and not some other person with the same name. To this end we use HERD - an entity recogniser and disambiguator (Södergren and Nugues, 2017) to annotate the corpora that we use as inputs in the program. It is the job of HERD to disambiguate which Göran Persson is meant in the sentence; whether in the above example Göran Persson refers to the former prime minister of Sweden with the id “Q53747” or the progg musician with the id “Q6042900”.

Features

The next step is to extract features from the examples found. We use a number of lexical and syntactic features. For each pair of entities found in the example sentences we extract a number of features:

- k words before the first entity in the sentence and their part-of-speech tags
- k words after the second entity in the sentence and their part-of-speech tags
- the sequence of words between the two entities and their part-of-speech tags
- the dependency path between the entities
- the dependency window for each of the two entities. The dependency window is a single dependency edge from an entity that is not part of the dependency path between the two entities
- the named entity tags of the two entities (e.g. Person/Location/Organisation etc)

The window size k has been varied during development to investigate the impact of different values of k. Currently we use a window size of 3 words.

The named entity tags can be either “Person”, “Location”, “Organisation”, “Miscellaneous” or “None”. The rationale behind this feature is that the tag can suggest whether it is likely a certain type of relation is expressed between the entities. For instance, a sentence with two entities of type “Person” is not likely to express the place of birth relation.

Furthermore, we use two syntactic features, the dependency path between the two entities, and for each entity a node that is not part of the dependency path. Syntactic features can be useful to differentiate between otherwise similarly expressed relations in sentences.

Feature Transformation

In order for a machine learning algorithm to learn it needs numeric input, thus we need to convert the words and their part-of-speech tags to some index.

One approach that we employ is **one-hot encoding**. The part-of-speech tags, the named entity tags, and the dependency features are all encoded in this way. This is suitable because the vocabulary size is rather small for these types of features; there is only a fixed number of part-of-speech classes.

However, for the word features, i.e. the word windows, we use Word2vec.

Chapter 4
Implementation

We constructed a system, that we refer to as `Prometheus` that consists of several software artefacts:

- Prometheus is the main program and is used to train the model and to use the aforementioned model to perform relation extractions; it implements our NLP pipeline.
- a proof-of-concept system to showcase Prometheus in a more user-accessible way, built for Sony and consisting of:
  - Prometheus Fact Checker, see Section 4.2, a backend to check facts against a database.
  - Prometheus Chrome Plugin, see Section 4.3, a browser plugin for Chrome that sends the currently viewed page in the browser to Prometheus Fact Checker and displays any found relations on the page and whether they could be verified or not by the backend.

4.1 Prometheus

Prometheus is the main result of this thesis and it is used to produce a trained model and to perform predictions, i.e. extract relations from the input corpus. Prometheus can perform extractions by either extracting relations from a given input corpus or by responding to HTTP requests with its built-in HTTP server. This can be useful for demonstration purposes and to provide a sort of fact-checking service, see Section 4.2 Prometheus Fact Checker.

In Figure 4.1 the packages in the program are laid out. As can be seen, there is one root package dubbed `com.sony.prometheus` which includes four sub packages:

- `stages` - includes implementations of all of the stages in the pipeline as well as the abstract traits that defines a stage

- `annotators` - includes the VildeAnnotator that can be used to send text to the Vilde machine at LTH which runs the Langforia language processing pipeline
- `utils` - includes some utilities for use throughout the program
- `interfaces` - defines the HTTP server that serves a REST:ful api that can be POST:ed to in order to extract relations from text

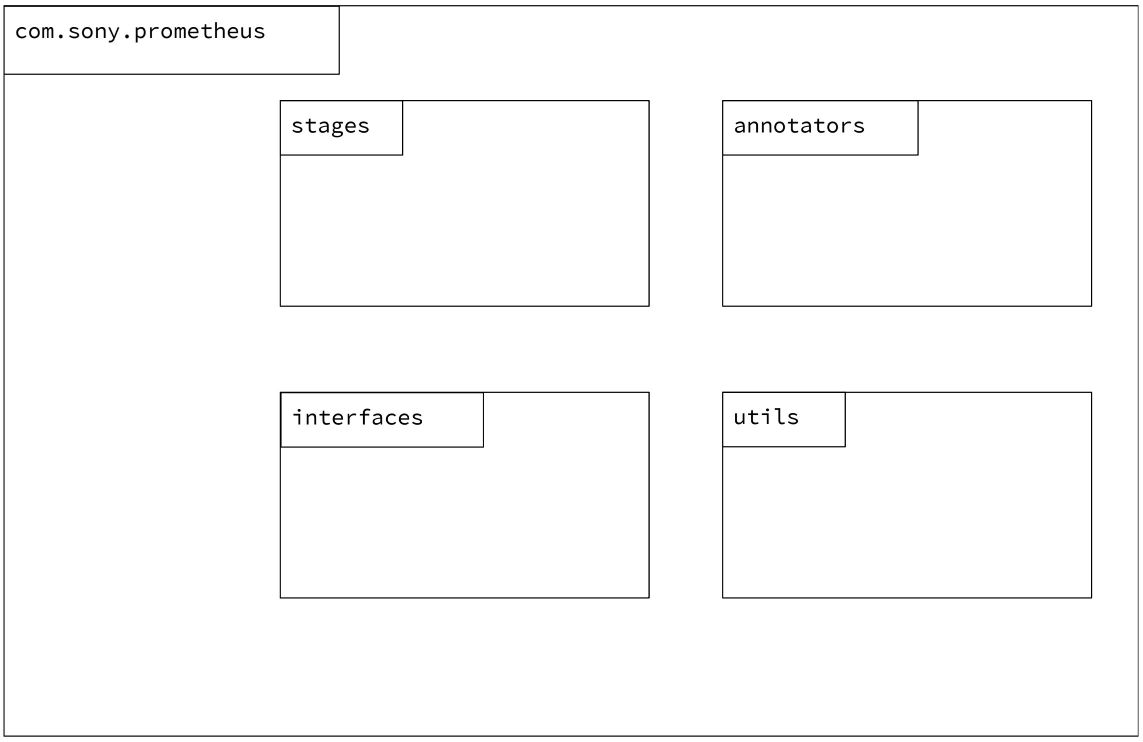


Figure 4.1: Overview of the different packages in the program

4.1.1 Architecture

Prometheus is written in Scala, has a monolithic architecture, and is designed to be able to execute on a cluster. We use Apache Spark extensively to be able to run Prometheus on a cluster in order to process large input corpora.

The general idea of the software architecture is to implement the pipeline in so called stages. A stage is a step in the pipeline and its output is some data that is saved to disk, this data can in turn be used as the input of the next step in the pipeline. This design ensures a data-centric way of performing the execution of the pipeline and enables very useful features.

The data is used as an indication of what actually needs to be computed. For instance, in order to extract features, **training sentences** are needed. If they already exist on disk, these will be used, but if they have been deleted they will be recomputed and saved to disk. This means that we have a caching mechanism built into the program. Any stage that is requested by a future stage will first check whether its output data already exists in which case this data will be returned or else this data will be computed and the output path returned, ready to be consumed by the next stage in the pipeline. If the data to be computed requires other stages, these will in turn recompute or use cached data.

The pipeline abstraction is programmatically represented in the file “Pipeline.scala” in the `com.sony.prometheus.stages` package. It defines two traits, `Data` and `Task`. A stage must at least mix in the Data trait but most stages also mix in the Task trait. Thus, there are no stages only implementing the Task trait. Code Listing 2 shows the structure of a typical stage in our pipeline.

```scala
class ExampleStage(path: String) extends Task with Data {
    override def getData(): String = {
        if (!pathExists(path)) {
          run()
        }
        path
    }

    override def run(): Unit = {
        // produce the data and save to 'path'
    }

    def readData(path: String): Seq[String] = {
        val data = ... // read data from disk
        data // returns the data
    }
}
```

Listing 2: Example stage implementation

Stages that are runnable mix in the `Task` trait which requires an implementation of the `run` method. When called it performs the computation but returns `Unit`. Stages where you can actually retrieve data from would also mix in the Data trait and would thus need to implement the `getData` method. This method returns the path on disk to the data produced by the stage if the data exists, otherwise, if it also implements the Task trait, `run` is called to produce the data.

In conclusion, every stage mixes in the Data trait and one can call `getData` to get the path to the data. In addition, some stages (most in fact), also mix in the Task trait and thus can themselves also produce the data if it is missing. And if it is missing, the stages will themselves call their `run` method to produce the data and then return the path to the data.

For other stages to consume the data, they will need to be able to read it as intended and to this end the stage that writes the data to disk will also have a method to read the data into memory and return a suitable data structure representing it.

An overview of the most important stages required to produce the trained model can be seen in Figure 4.2. The following sections will detail the stages that are involved in the execution of the pipeline.

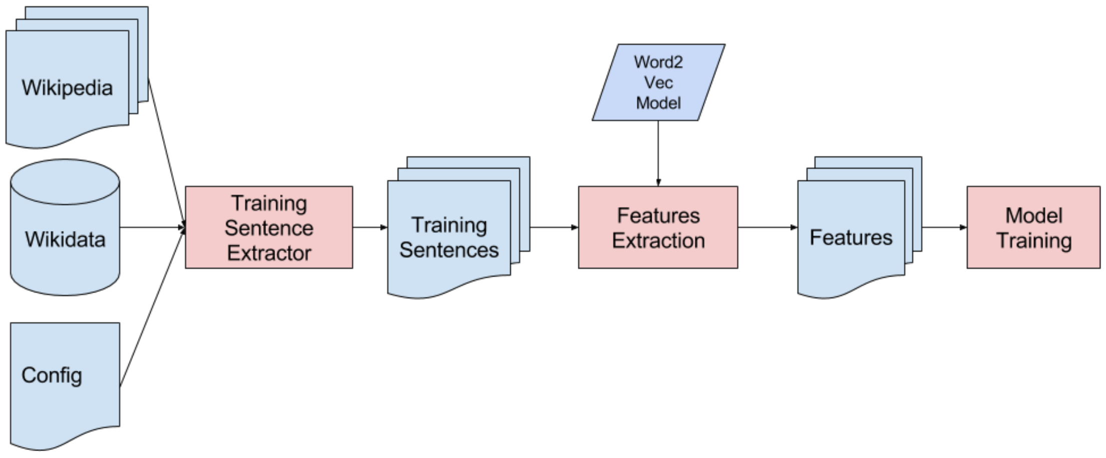


Figure 4.2: General overview of the model training stages in the pipeline

4.1.2 Corpus Reader

The Corpus Reader's purpose is to read the annotated corpus into memory. This stage is represented by the "Wikipedia" part of Figure 4.2. The corpus is read from **parquet** files, a columnar storage format commonly used by Apache Spark The stage also supports sampling a fraction of the corpus which can be convenient when one wants to quickly run through the pipeline but is not concerned with using the entire corpus.

We use a HERD-annotated Wikipedia dump as the input corpus (Södergren and Nugues, 2017). When run over a corpus it produces an annotated dump of the corpus. The annotations are stored in layers in Docforia. Docforia is an efficient multilayer document model that is designed to support processing of millions of documents and billions of tokens from large corpora such as Wikipedia (Klang and Nugues, 2016a). It also has support for cluster computing frameworks such as Hadoop and Spark.

Thus Corpus Reader requires that the annotated corpus is available in the working directory set by the main method in the program. If the corpus is not present, the Corpus Reader will throw an exception as it is not a Task stage and cannot compute its own output.

Entity Pair Extractor

The purpose of the Entity Pair Extractor is to extract "entity pairs"; pairs of entities that participate in a given relation. This constitutes the distant supervision. This module requires a Wikidata dump and a configuration object that describes which relations to extract entity pairs for. This module is thus represented by the "Wikidata" part of Figure 4.2. The Wikidata dump is read and all entities' properties are checked to see if they include a target relation. If a property matches a target relation, that Wikidata entity and the property's key constitute an entity pair for that specific relation. This module also defines the Relation data holder, which is a Scala case class. Apart from the sequence of entity pairs, a Relation contains the name of the relation, a string identifier for the relation (the Wikidata property identifier) and the expected types of the subject and object in the relation. For each relation there is an expected type tuple; i.e. for the married relation, we expect both the subject and object to be persons, and for the place of birth relation, we expect the subject in the relation to be a person while the object should be a location.

Training Data Extractor

The Training Data Extractor handles the extraction of training sentences. Its input is the corpus that the model should be trained on represented by the Corpus Reader as a Data stage, and the sequence of relations produces by the Entity Pair Extractor as described previously. The corpus is read sentence by sentence and the named entities in each sentence are matched against the known entity pairs. If there is a match, the matching pair and the sentence are returned as a positive training example.

Furthermore, if there are more than two named entities in the sentence, all other permutations of pairs of entities in the sentence are also returned but as negative training examples. Thus, from the same source sentence we extract as many examples as there are permutations of pairs of named entity pairs. See Figure 4.3 and Table 4.1 for an example of this. The Training Data Extractor also extracts random sentences that contain any entity pair that do not participate in any known relations. These are treated as negative examples.

Feature Extractor

The Feature Extractor stage extracts features from the sentences provided by the Training Data Extractor. Since the Training Data Extractor labels all permutations of entity pairs in a sentence as positive if at least one of the entity pairs is known to partake in a relation, all of the training sentences are not guaranteed to actually be positive. One of these permutations is an example of a positive example and the others are treated as “near-positive”. Thus, the Feature Extractor makes another distinction between the negative examples: *pure* negative or “near-positive”. Thus near-positive examples are examples extracted from a sentence which states the relation but which contain other entity pairs than the pair that actually participates in the relation.

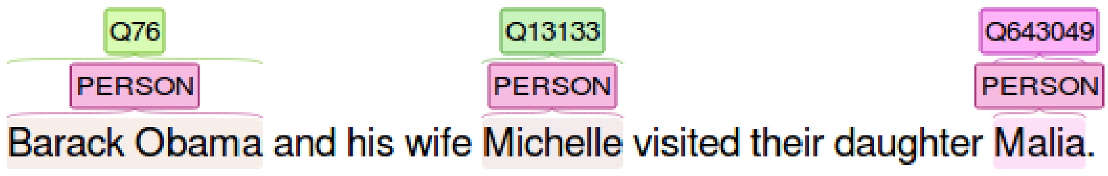

Q76
PERSON
Q13133
PERSON
Q643049
PERSON
Barack Obama and his wife Michelle visited their daughter Malia.

Figure 4.3: An example sentence stating a married relation.

An example is seen in Figure 4.3. In that sentence there are three named (and disambiguated) entities; Barack, Michelle and Malia Obama. The sentence states that Barack is married to Michelle and thus the positive example for the “spouse” relation would be the sentence together with the named entity pair consisting of Barack and Michelle. However, the Training Data Extractor would also have extracted the example with the named entity pairs (Barack, Malia) and (Michelle, Malia) even though the sentence do not state that they are married. The example is however very close to a positive example since there is a statement of marriage in the sentence, just not with that entity pair. These types of examples are distinguished by the Feature Extractor and labelled **near-positive**, meaning they are negative examples but ones that are hard to distinguish from the real positive examples. See Table 4.1 for the different types of training examples that would be extracted from the sentence.

Entity pair	Example type
(Barack, Michelle)	positive
(Barack, Malia)	near-positive (negative)
(Michelle, Malia)	near-positive (negative)

Table 4.1: Training examples extracted from the sentence: "Barack Obama and his wife Michelle visited their daughter Malia".

The reason we do this is to be able to balance the training examples into a good mix of positive, negative but near-positive examples, and purely random sentences. The idea is that the model should be able to easier disregard random sentences as negative but also have enough near-positive examples to be able to distinguish between tricky cases such as in the above example.

Apart from labelling the examples further, the Feature Extractor also actually extracts features. The features extracted are described in the Features section. Each example is returned as a "TrainingDataPoint", this data holder includes some attributes about the relation, i.e. name and identifier, the features extracted, the label, and also a Boolean flag `ent1IsSubject` indicating whether the first entity in the entity pair is the subject or not. All of the features extracted are either simple strings or sequences of strings. Thus there is no feature encoding happening in this stage. That is handled by the next stage, the Feature Transformer.

4.1.3 Feature Transformer

The Feature Transformer stage mix in both the Data and the Task stage and the output is the features transformed, or encoded, into their numerical representations to be consumed by the model training stages. This stage has a number of dependencies; apart from the Feature Extractor stage, which it requires in order to gain access to the features to transform, it also depends on a number of feature encoders.

Different feature encoders are used for different types of features as described in the Feature Transformation section. Those different types of encoders are also all implemented as stages:

- `PosEncoderStage`
- `NETypeEncoderStage`
- `DependencyEncoderStage`

For details of the implementation of these stages we refer to the source code (Gärtner and Larsson, 2017a).

4.1.4 Model Training

At this stage in the pipeline, the data pre-processing is complete and the models are ready to be trained. We train two models, one model is used for filtering and one for classification. The filtering model is a logistic regression model which is trained to discern between sentences that contain a relation and those that do not; it is a binary classification model. The purpose of this step is to act as a filter so that the input data to the classification model is cleaner.

The filter model is implemented as stage in the class `FilterModelStage` to be found in `FilterModel.scala` in the `com.sony.stages` package. The stage depends on the previously described Feature Transformer stage. It provides a method that trains the model and outputs some classification metrics that are useful to indicate how well the model performs. The examples are split into two sets, a train set and a test set in order to produce the metrics. Logistic regression computes a probability of each example to belong to either class. This lets us set a threshold that can be used to tune how much irrelevant sentences we want to discard at the cost of possibly excluding relevant training data.

For the actual implementation of logistic regression we use the built-in Spark implementation of Logistic Regression called `LogisticRegressionWithLBFGS` from Spark's `mllib` (The Apache Software Foundation, 2017c). See Section 3.1.8 for more details.

The model is automatically saved and thus when training is complete subsequent runs of the pipeline can load this model in from disk and does not need to retrain the model every time.

`ClassificationModelStage` is implemented in the `stages` package. Like the filter model it depends on the Feature Transformer stage to load in the data in the form of training sentences. Note that this means that the classification model has no dependency on the filter model. So in training the filter model and classification models are trained independently and on the entire dataset. The filter model's output is used only when performing predictions - the data fed into the classification model is only then filtered through the filter model to discard non-relation sentences. The classification is implemented as a fully-connected 3-layer neural network. The first layer has an input size equal to the length of the features and an output size of 512. The second layer has an input size of 512 and output size of 256 which naturally is the input size of the third and last layer. The output size of the last layer, and thus the whole network, is equal to the number of relation classes, including the null-class.

The backpropagation algorithm used is the stochastic gradient descent method, **ADAM** (Kingma and Ba, 2014) and the activation function is a **Rectified Linear Unit**, see Equation 4.1.

$$f(x) = max(0, x) \tag{4.1}$$

The network is built with the Java library "Deep Learning for Java" which is a an open source distributed deep learning library for the JVM (Deeplearning4j Development

Relation class	Size (English)	Size (Swedish)
Spouse	184,238	23,954
Place of birth	389,000	52,315
Educated at	69.068	1,064

Table 4.2: Distribution over the sizes of the training data for the different relation classes.

Team, 2017). A relevant related class is the `RelationModel` class which combines the filter model and the classification model into one model and provides methods to perform predictions.

Since the raw training data is quite unbalanced; some relations are easier to find examples of in the training corpus than others, a balancing method is needed. Otherwise the classification model might become biased towards classifying relations as the relation that has the most training data. This method is called `balanceData` and is implemented as a static method in the `RelationModel` companion object. It can either under- or over-sample the largest or the smallest class to the size of the smallest or largest class, respectively. The result is a training dataset with equally sized classes. In Table 4.2 we see the distribution over the sizes of the different relations. As can be seen, there is a considerable difference between the size of training data for the *Educated at* relation and the *Place of birth* relation.

There are three ways of utilising the model in the program. Via either the REST api, the evaluation methods in the `evaluation` package or via the `PredictorStage`.

4.1.5 Predictor Stage

The Predictor stage mixes in both Data and Task and it depends on a `CorpusData`, a `RelationModel` and feature encoders. Its purpose is to extract all relations it can find in the CorpusData with the help of the predictions it receives by feeding the corpus' sentences into the relation model. It returns a sequence of extracted relations that each contain the predicted subject, object and predicate as well as the source sentence and both the individual probabilities given by the filter model and the classification model and the combined probability of the model.

4.1.6 REST API

There is a module called `REST` residing in the package `interfaces` that launches an HTTP server that exposes an API in the form of a POST endpoint; `/api/<lang>/-extract`.

The endpoint expects requests with a data payload in the form of plain text that Prometheus will attempt to extract relations from. To perform the extractions, the `Predictor` class that is also used by the `PredictorStage` is sent the input text. However, the input to the Predictor needs to be pre-processed. To this end, the `VildeAnnotator` in the `stages` package is used. It sends the input text for annotation to the LTH machine called *Vilde* which is running the so-called Langforia pipeline (Klang and Nugues, 2016b), at `cs.vilde.lth.se:9000`. Vilde is set up to serve various NLP pipelines and the configuration we use is called "herd". The "herd" configuration runs CoreNLP 3.8.0 and HERD on the input text, responding with an annotated Docforia JSON response. The response is converted to a Docforia Document which now contains several layers of annotations containing named entities, named entity disambiguations, tokens, sentences, part-of-speech tags etc. The document now contains all the layers required for the Predictor to be able to extract features and perform extractions.

The response will be in JSON format and contain all extracted relations together with some meta data, see Listing 3 on page 59 for an example response.

Coreference Propagator

In the `utils` package there is an object called `Coref` with a single method: `propagate-Corefs`. The method takes one parameter in the form of a Docforia Document: `doc`, and returns `Unit`. It tries to resolve any coreference chains in `doc` by copying over the named entities of the antecedents to all of their anaphora. See Figure 4.4, where the antecedent, *Barack Obama*, previously annotated with the named entity, of type *PERSON* and the disambiguation *Q76* is copied to the anaphora; *He*. This effectively means that when this method has been applied to `doc`, it can be read as "Barack Obama is the president. *Barack Obama* is married to Michelle Obama".

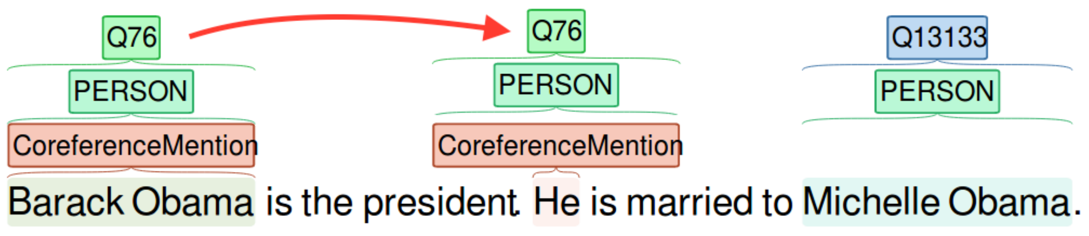


Figure 4.4: Coreference propagation

This module is be used by the REST API if the appropriate command line interface parameter is set when running Prometheus. Using this flag when running the REST API ensures that more sentences can be extracted from.

4.2 Prometheus Fact Checker

During the course of the project we also built a small proof-of-concept demonstration; the Prometheus Fact Checker (Gärtner and Larsson, 2017c). Refer to Figure 4.5 for an overview of the architecture of the system. As can be seen, the system consists of three main parts, the Prometheus main program running in "demonstration mode", the Prometheus Fact Checker backend server, and a client, in the form of a Chrome plugin.

The Prometheus Fact Checker backend exposes an API in the form of an HTTP POST endpoint; `/check`. Clients will send POST requests to this address with a URL payload and the backend will fetch the HTML page that resides on the URL posted, see steps 1 and 2 in the figure.

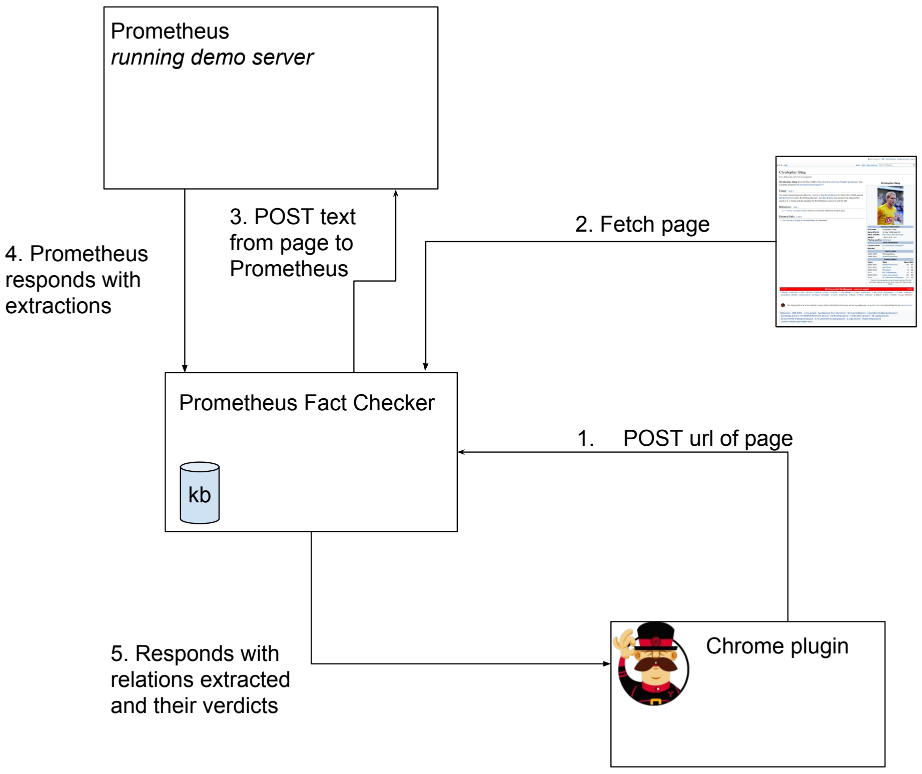

Prometheus
running demo server
4. Prometheus responds with extractions
3. POST text from page to Prometheus
2. Fetch page
Prometheus Fact Checker
kb
1. POST url of page
5. Responds with relations extracted and their verdicts
Chrome plugin

Figure 4.5: Prometheus Fact Checker

The backend will perform some processing of the HTML document and extract all paragraphs, i.e. the HTML `<p></p>` tags, on the page as plain text. The text will be split into chunks, where each chunk is up to 10 paragraphs large and up to 10 chunks will be sent in parallel to a running instance of Prometheus, configured to run in so-called demonstration mode; step 3 in the figure. In step 4, Prometheus will respond with any found relations in the chunks sent to it, see Listing 5 on page 61 for how this will look. Then the Prometheus' Fact Checker will compare the extractions from Prometheus with its knowledge base. The knowledge base consists of JSON files of previously done extractions performed by Prometheus over some corpus, e.g. the whole of the English Wikipedia.

For each extraction, the backend will try to find a match in the knowledge base and then label the extraction as either unknown, verified or disputed. If, for instance, the sentence:

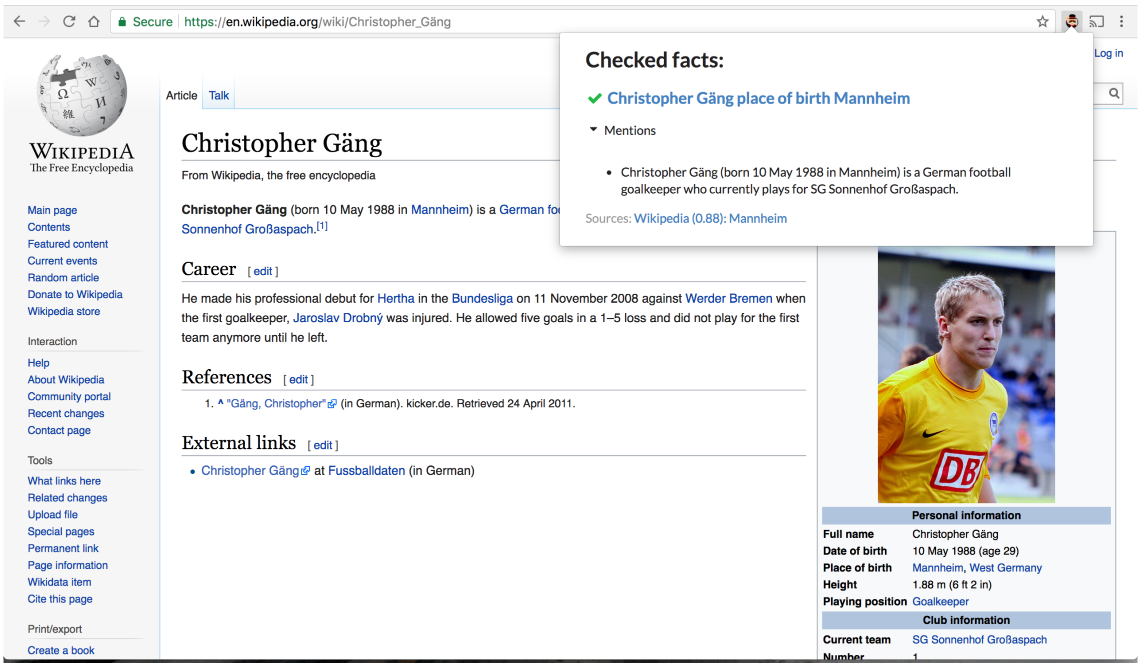
Jay-Z is married to American R&B singer Beyoncé.

is found on the page that the URL points to, Prometheus may extract the relation triple (Jay-Z, married, Beyoncé). If that exact same relation triple can be found in the knowledge base, the triple is labelled "verified", since, according to the knowledge base, it is a correct statement. If on the other hand a relation triple stating for instance (Jay-Z, married, Taylor Swift) is found, then a partial match on the subject and predicate but with a differing object is found (the object will be Beyoncé in the knowledge base). The resulting label in that case would be "conflicting" - indicating that the statement could be false. That label will be returned in the response together with the evidence, the disputing facts in the knowledge base.

4.3 Prometheus Chrome Plugin

As a more user friendly way of interacting with the Fact Checker, we developed a browser plugin for Chrome, called the Prometheus Chrome Plugin (Gärtner and Larsson, 2017b). See Figure 4.6 for a look at the interface.

The plugin works by extracting the URL of the website the user visits and sends an HTTP POST request to the backend, the Prometheus Fact Checker, with the URL as the data payload. It then awaits the response from the backend, displaying a progress animation in the meantime. If there is a non-empty response it presents the findings by listing all the statements that were found on the page together with their source sentences from the page and their verdict.



Checked facts:

✓ Christopher Gäng place of birth Mannheim

Mentions

- Christopher Gäng (born 10 May 1988 in Mannheim) is a German football goalkeeper who currently plays for SG Sonnenhof Großaspach.

Sources: Wikipedia (0.88): Mannheim

Christopher Gäng

From Wikipedia, the free encyclopedia

Christopher Gäng (born 10 May 1988 in Mannheim) is a German fo
Sonnenhof Großaspach.[1]

Career [edit]

He made his professional debut for Hertha in the Bundesliga on 11 November 2008 against Werder Bremen when the first goalkeeper, Jaroslav Drobný was injured. He allowed five goals in a 1–5 loss and did not play for the first team anymore until he left.

References [edit]

1. ^ "Gäng, Christopher" (in German). kicker.de. Retrieved 24 April 2011.

External links [edit]

- Christopher Gäng at Fussballdaten (in German)

Personal information	
Full name	Christopher Gäng
Date of birth	10 May 1988 (age 29)
Place of birth	Mannheim, West Germany
Height	1.88 m (6 ft 2 in)
Playing position	Goalkeeper
Club information	
Current team	SG Sonnenhof Großaspach
Number	1

Figure 4.6: Prometheus Chrome Plugin

Chapter 5

Evaluation

In this chapter we present our findings and discuss the implications of them. Finally we discuss possible future work on the system.

5.1 Experimental Setup

Prometheus' entire pipeline (data preprocessing, training and evaluation) can be run on LTH's natural language processing cluster, *Semantica*. It is a small Spark cluster consisting of about ten computers with around 20 GB of memory available to each worker. The pipeline can also be run on Amazon Web Services to perform multiple tests simultaneously.

The evaluation of the system is done in several steps. Firstly, during the training of the machine learning models we perform cross-validation to verify whether or not the models improves with each epoch. The result of the cross-validations however are not interesting since that evaluation is performed on parts of the training data. The similarity of the test data and training data makes the result biased since high scores can simply imply that the model over fitted to the training data.

In order to get a less biased evaluation of the system we complement the cross-validation with three more methods detailed below.

Using these methods we can tune the system's main hyperparameter "threshold" which is the threshold for the probability of a prediction being considered credible. For example the system might claim that the sentence "Barack Obama is married to Michelle Obama" would be represented as (Barack Obama, married to, Michelle Obama) with a certainty of 90%, but unless our threshold is lower or equal to 0.9 it will be discarded.

5.1.1 Google Dataset

In 2013 Dave Orr, Product Manager of Google Research, released a blog post containing several human-evaluated corpora containing relation extractions from Wikipedia (Google Research, 2013). Each extraction was evaluated by at least five humans labelling the extraction as either correct or incorrect. The dataset contains around 10,000 examples of *place of birth* relations and over 40,000 of *attended or graduated from* relations. The examples contain small text snippets called "evidences" from which the system is supposed to extract the relation. Each example also comes with a relation triple that may be correct or incorrect according to the human judges. See Listing 4 for an example from the Google dataset. Note that the Google Dataset is only available in English, as such this evaluation is only available for English.

In the original format the Google dataset uses Freebase IDs for the subject, object and predicate. Since Freebase is defunct we translate these IDs to Wikidata IDs. This conversion process is not successful for every example so those which cannot be converted are discarded.

Next we filter out all examples judged incorrect by the judges. The reason for this is that the correct examples can be used to calculate recall but the incorrect examples are harder. Even if our system could extract the correct relation in the evidence we would have nothing to compare against.

After all filtering we have about 1,600 *place of birth* relations and about 3,800 "attended or graduated from" relations.

In the Mintz et al. (2009) paper they evaluate the top 100 and top 1,000 most probable extractions according to the model itself. In order to compare results we also calculate the scores for the best 100 extractions.

5.1.2 Manual Evaluation

Furthermore we perform a manual evaluation by running the extractor over the entire English Wikipedia and manually evaluating the top 100 extractions. This was again used as a measure to compare against the Mintz et al. results.

5.1.3 Wikidata Evaluation

Lastly we use the extraction from running the extractor on Wikipedia. These extractions could be seen as a knowledge graph like Wikidata.

We calculate some basic statistics such as: the number of facts in our database, how many of "our" facts that are present in Wikidata, and how many facts are conflicting. We defined conflicting as:

1. Given an extraction (SubjectA, PredicateP, ObjectB).

2. It is deemed conflicting iif there exists one or more other triples containing the same subject and predicate but never the same object. Such as (SubjectA, PredicateP, ObjectC).

This definition of conflict above follows the **local closed-world assumption** defined in Dong et al. (2014b). That conflicts occur when the knowledge base has some triples containing (Subject, PredicateP) but none of them contains the object ObjectB.

So for the "education at" relation where multiple institutions are possible, a conflict occurs if we have some information about a subject's education but the extracted fact is not part of the knowledge base. Where as if we have no knowledge we don't label this fact conflicting.

The Wikidata evaluation is mainly used to provide a lower bound for the precision of system. Since we do not know how many facts in Wikidata are present in Wikipedia and vice versa the returned precision is only a lower bound.

5.2 Results

This section contains the final result for the system for our three evaluation types. All results are from the same model, from the same training sessions and with the same hyperparameters.

The relations evaluated are listed in Table 3.3.

5.2.1 Google Dataset

In Table 5.1 the final evaluation scores on the Google dataset are displayed. Note that the Google dataset was only available in English. **HERD Recall** is the recall of the underlying named entity disambiguation software on the dataset. This provides the theoretical upper bound for our recall. Table 5.2 shows the result of the subset of exampels where HERD successfully found the named entities.

Relation	Data points	True positives	Recall	Precision	F1	HERD Recall
Place of birth	1616	489	0.30	0.71	0.42	0.46
Educated at	3850	206	0.054	0.22	0.09	0.21

Table 5.1: Results on the Google dataset.

Relation	Data points	True positives	Recall
Place of birth	756	489	0.65
Educated at	791	71	0.07

Table 5.2: Results on the Google dataset on the subset of examples where HERD successfully disambiguated the named entities.

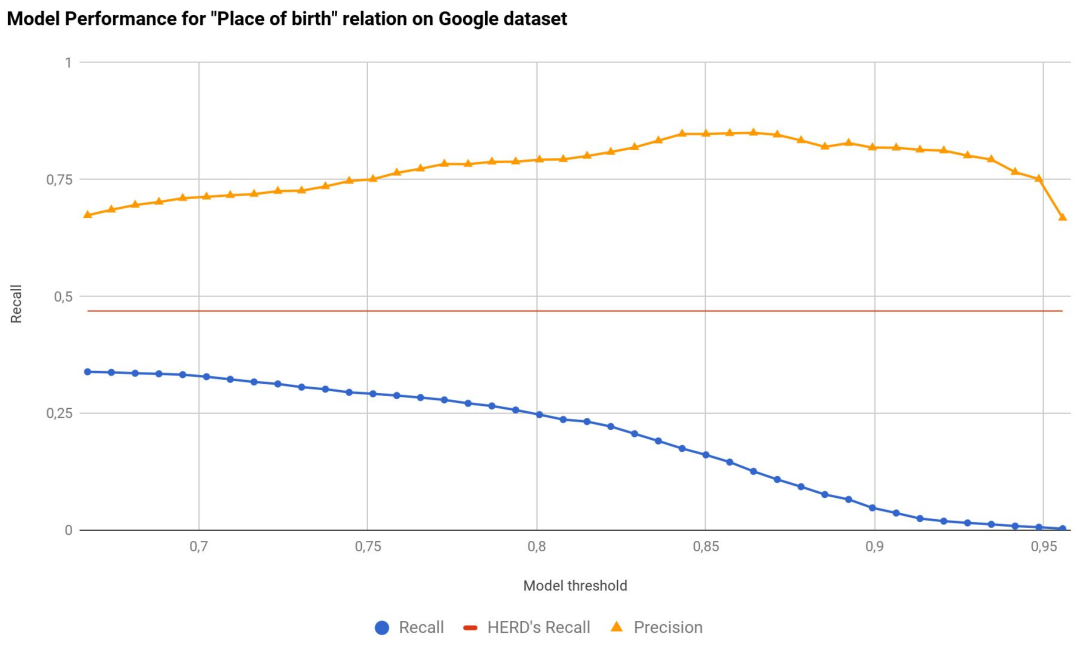


Figure 5.1: The figure shows the results of the Google dataset evaluation for the "Place of birth" relation for varying model threshold.

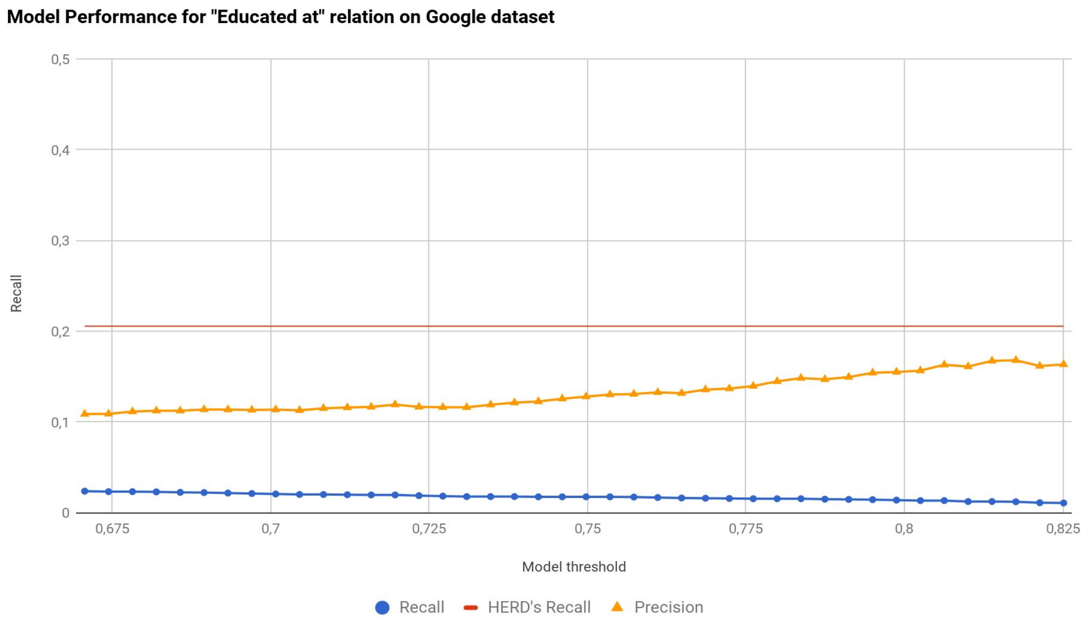


Figure 5.2: The figure shows the results of the Google dataset evaluation for the "Educated at" relation for varying model threshold.

5.2.2 Manual Evaluation

Table 5.3 shows the result from the manual evaluation on the top 100 (as scored by the model) extractions from Wikipedia. Note that for the *Educated at* relation in Swedish the system found only 7 relations altogether therefore we excluded it from the table.

Language	**Relation**	**Precision**	**Mintz et al. (2009) Top 100 Precision**
English	Place of birth	**0.89**	0.78
English	Spouse	0.32	-
English	Educated at	0.75	-
Swedish	Place of birth	0.96	-
Swedish	Spouse	0.79	-
Swedish	Educated at	-	-

Table 5.3: Result from the manual evaluation from extraction on Wikipedia.

5.2.3 Wikidata Evaluation

The results of the Wikidata evaluation are shown in Table 5.4 as well as in Figures 5.4, 5.5, 5.6, 5.7 and Figure 5.8.

The “verified rate” metric represents how large percentage of the extractions that are found in Wikidata. Looking at Figure 5.3 it is defined according to Equation 5.1.

$$\text{verified rate} = \frac{\text{verified extractions}}{\text{relation triples extracted}} \tag{5.1}$$

This metric provides a lower limit for our system’s precision on the dataset.

Language	**Relation**	**Extractions**	**Verified rate**
English	Place of birth	327859	0.43
English	Educated at	71231	0.19
English	Spouse	924723	0.045
Swedish	Place of birth	15786	0.64
Swedish	Educated at	0	-
Swedish	Spouse	516	0.56

Table 5.4: Result from the manual evaluation from extraction on Wikipedia with a model threshold of 0.75.

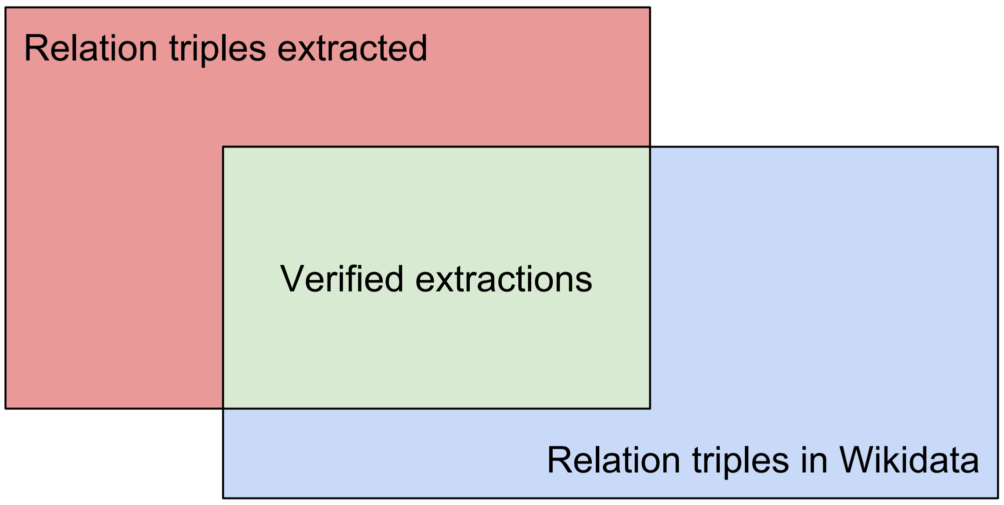


Figure 5.3: The figure explains how the verified extractions relates to Wikidata triples and extracted triples.

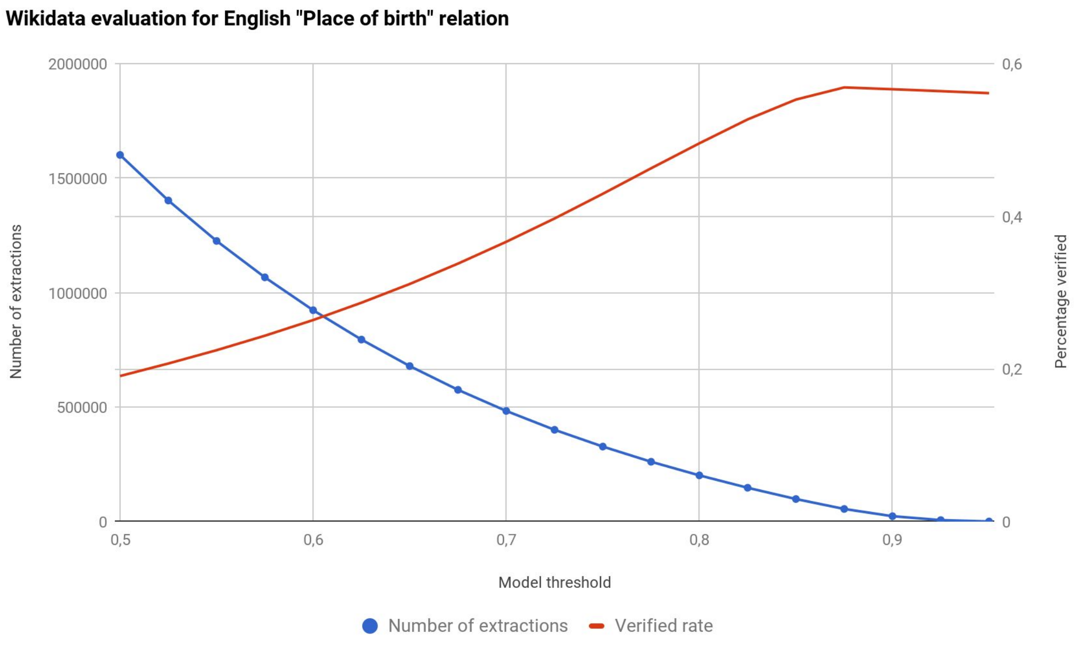


Figure 5.4: The figure shows the results of the Wikidata evaluation for the “Place of birth” relation in English for varying model threshold.

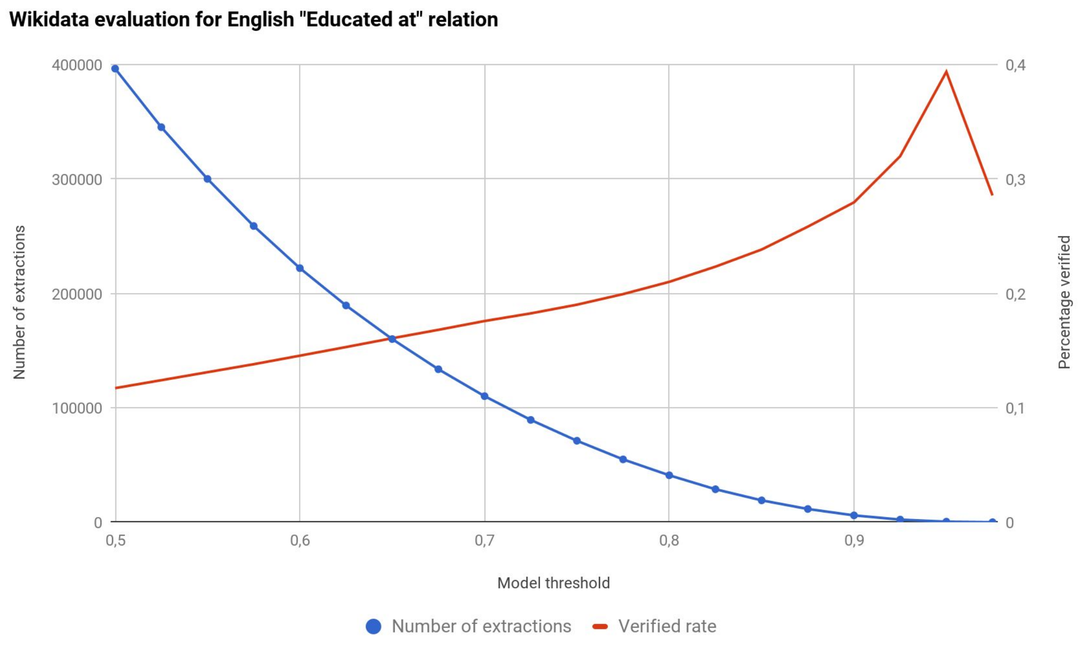


Figure 5.5: The figure shows the results of the Wikidata evaluation for the “Educated at” relation in English for varying model threshold.

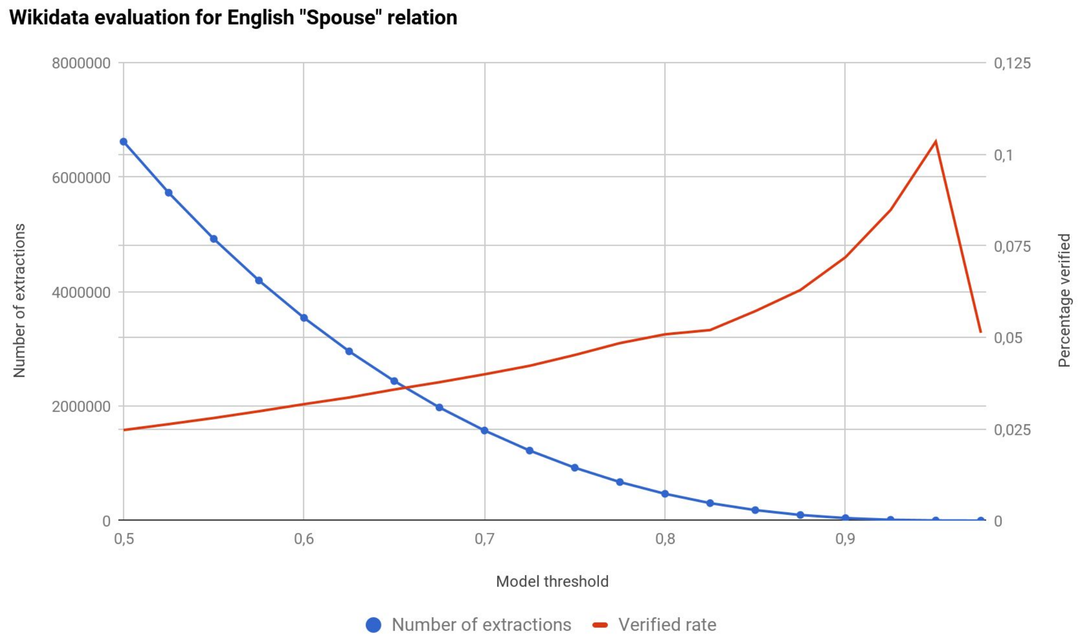


Figure 5.6: The figure shows the results of the Wikidata evaluation for the “Spouse” relation in English for varying model threshold.

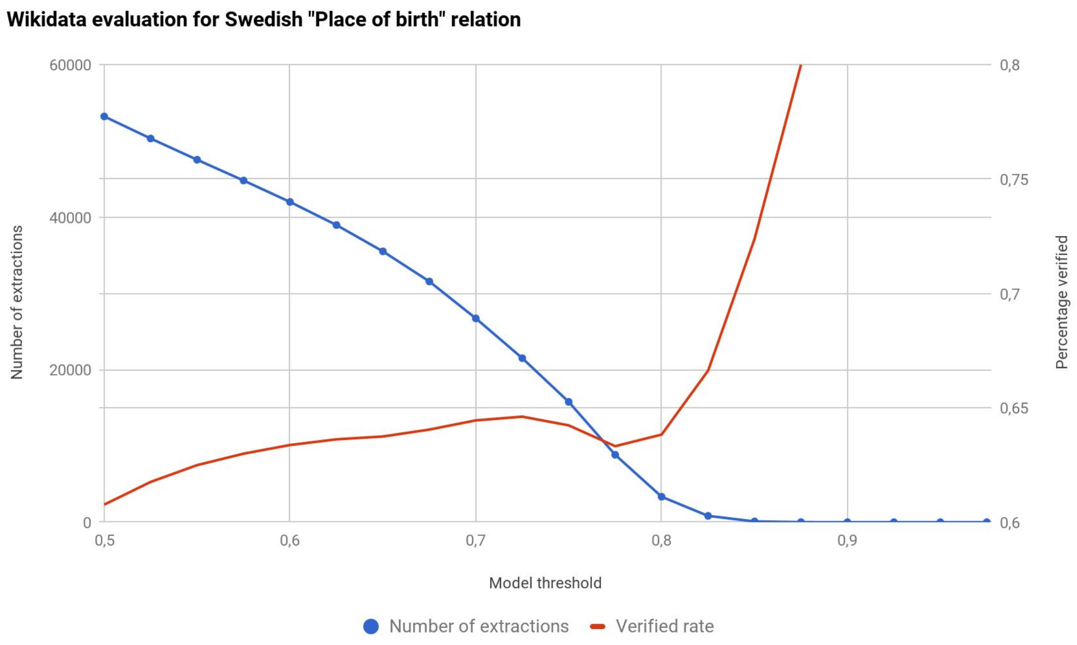


Figure 5.7: The figure shows the results of the Wikidata evaluation for the "Place of birth" relation in Swedish for varying model threshold.

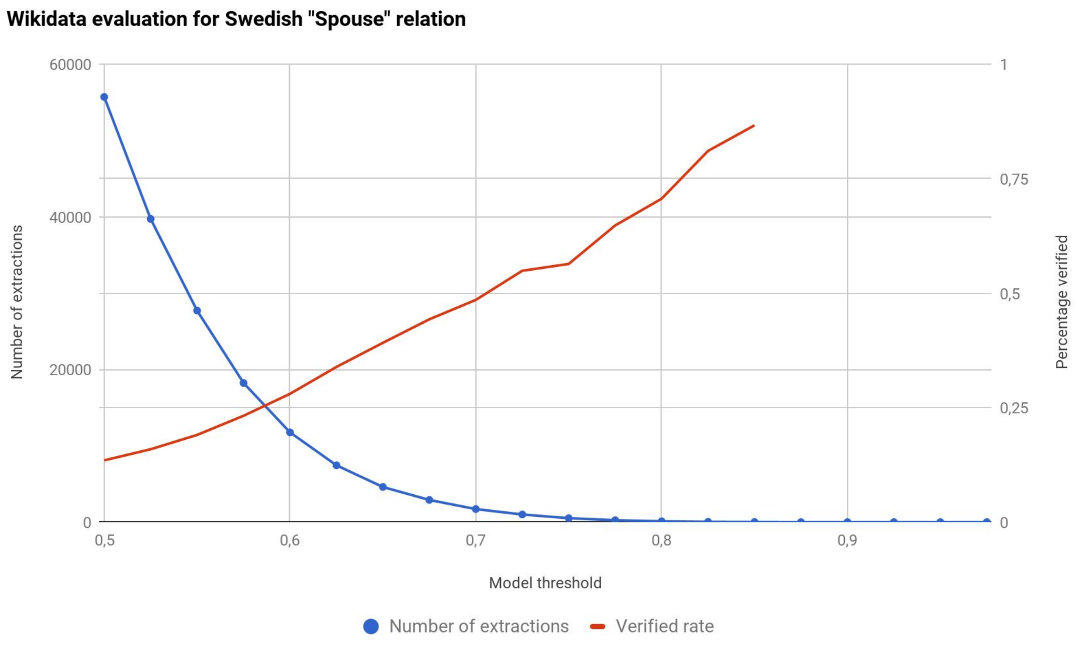


Figure 5.8: The figure shows the results of the Wikidata evaluation for the "Spouse" relation in Swedish for varying model threshold.

5.3 Discussion

5.3.1 Google Dataset

As seen in Table 5.1 the results for the *Place of birth* relation are quite promising while the *Educated at* relation is much worse. While investigating this discrepancy we found that the formulations in our training corpus, Wikipedia, differed greatly between relations. The articles on Wikipedia usually state certain types of relations in a very formulaic and encyclopedic way without a lot of variation, such as:

> Christopher Gäng (born 10 May 1988 in Mannheim)...

So the article starts with the name of the person, followed by the date and place of birth in parentheses. This type of formulation is not always used, and there are also examples where the place of birth is stated in a more “natural” way such as:

> Persson was born in Vingåker in Södermanland, Sweden, in a working-class home.

However, this illustrates the fact that using Wikipedia as our training corpus means that the model will associate certain formulations and grammatical constructs to a certain relation very strongly.

We use Wikipedia as our training corpora and there are several beneficial reasons for doing so. Wikipedia is free and open and multilingual and via Wikidata we can use the same set of entity pairs as our distant supervision for multiple language editions.

This type of constrained language will affect the model’s performance on datasets not using similar language, such as blog posts or news articles. To counter this type of overfitting it would be advisable to train the model on datasets from different sources. The reason we did not use other sources was because that would require those corpora to be annotated by HERD and the other NLP tools, such as CoreNLP, which is a slow process.

Furthermore the recall of our system is bounded by the performance of the underlying tools. Our pipeline is highly dependent on the previous parsing, annotation, and specifically the named entity recognition and disambiguation.

In order to be able to extract features we need the sentences to be correctly annotated with *disambiguated* named entities; otherwise there will be no facts to check against in the knowledge base. If for instance we would only extract the named entity “Göran Persson”, but not know which Göran Persson the mention refers to, it would be impossible to verify factual statements about that named entity because we would not be able to resolve it to an entity in the knowledge base. This means that if our underlying tools fails to disambiguate a named entity, or fail to detect a named entity altogether in the input sentence, our program will fail to extract any facts from that sentence. This is perhaps the biggest constraint of our system - the dependency on a named entity recogniser and disambiguator.

For the evaluation this means that HERD’s 46 % recall on the *Place of birth* relation becomes our recall’s upper bound. There is no way of getting around this constraint. As seen in Table 5.2 our recall is 65 % out of the correctly annotated examples.

As for the *Educated at* relation the HERD recall is even lower but that alone does not explain our poor results. A big reason seem to be that many examples in the dataset actually lack the full name of the person in question, like this sentence:

> He studied botany at the University of Leipzig under Christian Gottlieb Ludwig (1709–1773). In 1752 he succeeded Abraham Vater (1684–1751) as professor of botany and anatomy at the University of Wittenberg, where in 1782 became a professor of therapy

This is not only a coreference problem but actually the example completely lacks the named entity. This yields a low recall for the system. Thus the poor evaluation is also partly due to a hard and sometimes impossible evaluation set.

The biggest reason for the poor performance may stem from few training example for the *Educated at* relation as seen in Table 4.2. This is especially detrimental to the Swedish model. As seen in Table 5.3 the system does not even find 100 extractions from the entire Swedish Wikipedia.

5.3.2 Manual Evaluation

The manual evaluation, while small scale, is directly comparable to the Mintz et al. (2009) evaluation and can give us precision for the system on a real world dataset for both English and Swedish.

The English *Place of birth* relation in Table 5.3, which is the only relation that Mintz et al. (2009) also use, shows very good results and is a clear improvement. The *Place of birth* relation is also very good in Swedish. As discussed this is partly due to the very formulaic sentences in Wikipedia for this relation. Though it should be noted that since we only share one relation with the Mintz et al. (2009) system and they perform larger evaluation we cannot claim that our system outperforms their system in general. That would most likely require a top 1000 evaluation for their selected 10 relations.

The *Educated at* relation in English gives a precision of 75 %. It is interesting to note however, that of the 25 relations that were incorrect, 7 of these were due to incorrect named entity disambiguation. Out of the rest of the 18 incorrect extractions, an overwhelming majority were similar to this type of sentence:

> Judge Underhill teaches a course on Federal Courts as an adjunct professor at University of Connecticut School of Law, and a course on Federal Sentencing at the University of Virginia School of Law.

That is, sentences stating that a person holds a professorship or other academic position at a university, which is not the same thing as being *educated at* that same university. A few of the extractions also equate an honorary degree with a real degree – something counted as incorrect in the evaluation. In conclusion, the top 100 English extractions of *Educated at* are of high quality or at least very near being correct. As for the Swedish evaluation of *Educated at*, there were only 7 extractions made so these were not included in the table. It is unclear why the extractor found so few examples in Swedish but it is probably a combination of bad named entity disambiguation for Swedish regarding educational institutions and a lack of coreference resolution.

The *Spouse* relation showed very varied results between English and Swedish with the Swedish model vastly outperformning the English. In the Swedish extractions the top 100 extractions were almost entirely related to royal families where lineage was denoted in a formulaic and clear manner. This would have helped the model as was the case of the

Place of birth relation. The mistakes were usually due to convoluted sentences containing many entities such as mistresses, illegitimate children as well as the actual married couple.

5.3.3 Wikidata

The Wikidata evaluation is the largest of our evaluations since we try to use our extractor to extract facts from the entire Swedish and English Wikipedia. As can be seen in Table 5.4 the extractor performs very well on the English and Swedish *Place of Birth* relation. Again, this could be due to the extractor having learnt the very specific formulation in use on Wikipedia regarding this relation.

For the *Spouse* relation, we see that we have a very good verified rate for Swedish while it is considerably worse for English. Another interesting aspect is that for Swedish, we fail to extract any *Educated at* relations at all while for English the results are much better.

It is unclear what the underlying cause to these discrepancies may be, but note that a low verified rate does not mean that all of the extractions not verified are *incorrect* – it only means that they are not present to be verified in the Wikidata dump. As such it only provides us with a lower bound for the precision. We can consider *at least* the verified extractions as correct. There could also be a discrepancy between Wikipedia and Wikidata; Wikidata is supposed to represent a knowledge graph over Wikipedia but there are no guarantees that the Wikidata database is 100 % up to date with Wikipedia.

When doing the Wikidata evaluation we evaluated our model with varying threshold for the three relations for both English, and the two for Swedish (excluding *Educated at*). Overall, the graphs demonstrate the expected behaviour; a higher threshold yields a higher verified rate while the threshold is inversely proportional to the number of extractions. Thus there is a tradeoff between precision and recall as can be expected. We have settled on a threshold of 0.75 since that seems to give reasonably accurate extractions while still keeping a reasonable number of extractions.

It should be noted that the lower number of extractions for Swedish compared to English is to be expected since the Swedish Wikipedia has fewer articles.

5.3.4 General Discussion

The evaluation of the program carried out has shown that it is possible to implement a relation extractor using the distant supervision approach. Furthermore it is possible to achieve quite good precision for our extractor for certain relations.

Looking at the results of the various evaluations it is clear that most natural-language sentences are not composed in the way that would maximise the performance of our extractor. Since we train on a corpus without coreferences resolved, the amount of training data becomes limited to the type of sentences with proper mentions of the named entities. For example a sentence such as "Barack Obama married Michelle Obama" would occur in the training set as long as HERD successfully disambiguates Barack and Michelle Obama and these two share the *spouse* relation in the Wikidata dump; something that is likely the case. However, consider the sentence "He married Michelle Obama". This is the type of sentence that includes two mentions, but only one antecedent; *Michelle Obama*. The other is a so-called anaphor. Unless the anaphoric term can be resolved to its antecedent earlier in the text where it is stated what the *He* refers to, the system will not be able to extract any facts. Yet, these types of sentences are very common since most natural-language contain a lot of mentions that only abbreviate the antecedent, usually in the form of prononuns. Resolving these coreferences, e.g. *He, Barack Obama*, would be needed to be able to give the extractor a chance at extracting any facts from these types of sentences.

We have implemented a module to propagate any coreferences from their anaphora to their antecedents if the document in question has been annotated with the appropriate layers, see Section 3.1.5. However, doing coreference resolution is very costly computation-wise and our training corpus does not include coreferences. Furthermore, it is only available for English and not Swedish.

We have experimented with coreference resolution when doing evaluation on the Google dataset and there is support for switching it on for the REST API for demonstration purposes as well. However, we have noticed that using coreferences when evaluating the Google dataset results in considerably worse precision. A potential explanation for this could be that that the amount of training data will still be affected by the lack of coreference resolution and this will also bias the training set towards the type of sentences that do contain proper mentions. One could imagine that sentences with proper mentions are structured in a certain way and state the relations in a more formal manner than when pronomials are used. Thus, using coreferences in testing but not in training does not work well.

During development of our program we have from the very beginning have scalability as an important goal: being able to scale our pipeline with the number of relations in a sustainable way. The execution time of the program would need to grow significantly less than linear in the number of relations. Especially in pre-processing this is important; going over the entire training corpus more than once would not be feasible. Our pre-processing is efficiently implemented and scales well with the number of relations. However, for each relation we must produce training data. The training data is separate for each type of relation. This is a property of the problem domain, separate relations will in most cases be stated in separate sentences. There are of course sentences where a single sentence can contain more than one fact. However, these will still be treated as separate training examples to our program. Recall that a single sentence in our program is not a single example, rather it can contain a multitude of different training examples – one for each permutation of pairs of entities in the sentence.

This means that the more relations we support in our program, the more training data will be produced and thus the training will take longer. Producing less training data is not something that would be desirable, rather a different training algorithm that is faster could potentially be a solution if one wanted to improve performance when training the relation extractor for many relations.

Another approach could be to train multiple models for different types of relations in parallel and then combine the models when doing the extractions.

Furthermore, because our dataset is so large, we use a cluster to train the model. To enable training in parallel we use **parameter averaging**. While this speeds up computation it can impact model accuracy negatively (Su and Chen, 2015). Experiments to train a model on a single powerful computer could be done to investigate the effects of the parameter averaging. In case the effects are substantial the hyperparameters controlling the parameter averaging should be tweaked carefully to ensure minimal model degredation.

Finally, during the course of this thesis, we have both gained a lot of experience with technologies such as processing large amounts of data with the help of cluster computing. This has been a very rewarding experience and has taught us a lot, while it has forced us to apply our skills learnt during the course our education. We firmly believe that given more time this system could yield very good results but due to thesis time constraint we will limit ourselves to listing some of these improvements in the following section.

5.3.5 Future work

Since the result varies by relation and language it would be beneficial to have varying thresholds for each combination. This would allow the overall extractions to have a more even quality.

The current iteration of the system uses a relatively shallow network. Increasing the depth of the network, while requiring more computation, could possibly improve the performance of the model. In the field of computer vision there exists a trend of adding more layers, from around 16 in AlexNet (Krizhevsky et al., 2012) to more than 150 in ResNets (He et al., 2015) resulting in an increased performance. Current research by LeCunn et al. shows that there exists potential for using very deep networks in natural language processing as in computer vision (Conneau et al., 2016).

When it comes to neural network it is not only the depth of the network that is an important factor. Tweaking hyperparameters such as the number of neurons in each layer and which activation functions are contributing factors. Even more interesting would be to experiment with more complex layers than the simple feed-foward layers used by the current model. There exists promising research on using convolutional networks for natural language tasks (Conneau et al., 2016).

Furthermore it would be interesting to experiment with other features and feature representations, such as comparing the results of using GloVe word embeddings to using Word2vec embeddings (Pennington et al., 2014). Retraining and tweaking the word embeddings may also increase the model performance.

A common problem with using neural network in natural language processing is that neural networks require a fixed input size and sentences of course vary in length. There exists a new encoding format, FOFE, that projects sentences into a feature space of fixed dimension (Zhang et al., 2015). Experimentation with this format for the window features would be very interesting since it would allow for arbitrarily long windows.

Since the result of the program output is highly dependent on the quality and correctness of the underlying annotations on the input corpora, switching to another named entity recogniser and disambiguator with better precision and recall would automatically improve our results as well. This would probably not incur a significant amount of extra work either; as long as the new tool can produce Docforia-annotated documents the pipeline would not need to change.

Furthermore it should be noted that Deeplearning4j was quite problematic to setup for usage on Spark and since it uses off-heap memory for the matrix computations. That meant fine tuning the amount of memory allocated to the Spark process and to the Deeplearning4j process. The authors would look into the possibility of using another deep learning framework for future versions of the system.

Finally we ran all our training on CPUs since the Semantica cluster does not have GPUs. For further development it would be very good to use GPUs for the deep learning training to significantly speed up the training.

Chapter 6
Conclusion

This thesis has described the process of creating as well as the results of a multilingual relation extraction system.

Our results show that the system performs well for select relations, especially on input data with a very rigid structure. The system outperforms the Mintz et al. (2009) system on the single relation that they share.

For the future we recommend looking at another named entity disambiguation system, adding features such as FOFE encoding, as well as testing another deep learning framework. With these changes and further hyperparameter tweaking we firmly believe the system can yield even better results.

Bibliography

Bishop, C. M. (2009). *Pattern recognition and machine learning*. Springer.

Conneau, A., Schwenk, H., Barrault, L., and LeCun, Y. (2016). Very deep convolutional networks for natural language processing. *CoRR*, abs/1606.01781.

Dean, J. and Ghemawat, S. (2008). Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113.

Deeplearning4j Development Team (2017). Deeplearning4j: Open-source distributed deep learning for the jvm, apache software foundation license 2.0. `https://deeplearning4j.org/`. Accessed: 2017-08-07.

Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S., and Zhang, W. (2014a). Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 601–610, New York, NY, USA. ACM.

Dong, X. L., Gabrilovich, E., Heitz, G., Horn, W., Murphy, K., Sun, S., and Zhang, W. (2014b). From data fusion to knowledge fusion. *Proc. VLDB Endow.*, 7(10):881–892.

Google (2017). Freebase data dumps. `https://developers.google.com/freebase`. Accessed: 2017-08-07.

Google Research (2013). 50,000 lessons on how to read: a relation extraction corpus. `https://research.googleblog.com/2013/04/50000-lessons-on-how-to-read-relation.html`. Accessed: 2017-03-28.

Gärtner, E. and Larsson, A. (2017a). Prometheus. `https://github.com/Prometheus-Extractor/prometheus`. Accessed: 2017-08-26.

Gärtner, E. and Larsson, A. (2017b). Prometheus chrome plugin. `https://github.com/Prometheus-Extractor/prometheus-chrome-plugin`. Accessed: 2017-08-26.

Gärtner, E. and Larsson, A. (2017c). Prometheus fact checker. `https://github.com/Prometheus-Extractor/prometheus-fact-checker`. Accessed: 2017-08-26.

Hastie, T., Tibshirani, R., and Friedman, J. H. (2013). *The elements of statistical learning: data mining, inference, and prediction*. Springer.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

Holton, G. (2016). *Albert Einstein, historical and cultural perspectives*. Princeton University Pres.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Kirby, E. J. (2016). The city getting rich from fake news. *BBC News*. Accessed: 2017-08-10.

Klang, M. and Nugues, P. (2016a). Docforia: A multilayer document model. *Proceedings of SLTC*.

Klang, M. and Nugues, P. (2016b). Langforia: Language pipelines for annotating large collections of documents. In *COLING (Demos)*, pages 74–78.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

Mintz, M., Bills, S., Snow, R., and Jurafsky, D. (2009). Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2 - Volume 2*, ACL '09, pages 1003–1011, Stroudsburg, PA, USA. Association for Computational Linguistics.

Nugues, P. M. (2006). *An introduction to language processing with Perl and Prolog: an outline of theories, implementation, and application with special consideration of English, French, and German*. Springer.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Quirk, C. and Poon, H. (2016). Distant supervision for relation extraction beyond the sentence boundary. *CoRR*, abs/1609.04873.

Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition.

Su, H. and Chen, H. (2015). Experiments on parallel training of deep neural network using model averaging. *CoRR*, abs/1507.01239.

Södergren, A. and Nugues, P. (2017). A multilingual entity linker using pagerank and semantic graphs. In *Proceedings of the 21st Nordic Conference on Computational Linguistics, NoDaLiDa, 22-24 May 2017, Gothenburg, Sweden*, number 131, pages 87–95. Linköping University Electronic Press, Linköpings universitet.

The Apache Software Foundation (2017a). Apache hadoop. `http://hadoop.apache.org`. Accessed: 2017-08-08.

The Apache Software Foundation (2017b). Apache spark. `https://spark.apache.org/`. Accessed: 2017-08-07.

The Apache Software Foundation (2017c). Apache spark, linear methods, logistic regression. `https://spark.apache.org/docs/1.6.1/mllib-linear-methods.html#logistic-regression`. Accessed: 2017-08-07.

Timberg, C. (2016). Russian propaganda effort helped spread 'fake news' during election, experts say. *Washington Post*. Accessed: 2017-01-31.

W3C Foundation (2014). Resource description framework. `https://www.w3.org/RDF/`. Accessed: 2017-08-08.

Wikidata (2017a). Wikidata. `https://www.wikidata.org/wiki/Wikidata:Main_Page`. Accessed: 2017-04-03.

Wikidata (2017b). Wikidata. `https://www.wikidata.org/wiki/Wikidata:List_of_properties/all`. Accessed: 2017-04-03.

Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., and Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 2–2, Berkeley, CA, USA. USENIX Association.

Zeng, D., Liu, K., Chen, Y., and Zhao, J. (2015). Distant supervision for relation extraction via piecewise convolutional neural networks. In *EMNLP*, pages 1753–1762. The Association for Computational Linguistics.

Zhang, S., Jiang, H., Xu, M., Hou, J., and Dai, L.-R. (2015). The fixed-size ordinally-forgetting encoding method for neural network language models. In *Proceedings of ACL*, pages 495–500.

Appendices

```
[
  {
    "classificationProbability": 0.9999973773956299,
    "filterProbability": 0.8461826559000445,
    "obj": "Q2119",
    "predictedPredicate": "P19",
    "probability": 0.8461804366977133,
    "sentence": "Christopher Gäng (born 10 May 1988 in Mannheim) is a
                 German football goalkeeper who currently plays for
                 SG Sonnenhof Großaspach.",
    "source": "dynamic",
    "subject": "Q474877"
  }
]
```

Listing 3: Example response from Prometheus running in demo mode.

```
{
  "pred": "\/people\/person\/place_of_birth",
  "sub": "\/m\/02qvl5t",
  "obj": "\/m\/0345h",
  "evidences": [
    {
      "url": "http:\/\/en.wikipedia.org\/wiki\/Emmanuel_Scheffer",
      "snippet": "Emmanuel Scheffer (, born 1 February 1924 in
                  Germany) is an Israeli football coach."
    }
  ],
  "judgments": [
    {
      "rater": "16651790297630307764",
      "judgment": "yes"
    },
    {
      "rater": "2050861176556883424",
      "judgment": "yes"
    },
    {
      "rater": "1855142007844680025",
      "judgment": "yes"
    },
    {
      "rater": "11595942516201422884",
      "judgment": "yes"
    },
    {
      "rater": "16169597761094238409",
      "judgment": "yes"
    }
  ]
}
```

Listing 4: An example of a "place of birth" relation from the Google Dataset

```
[
  {
    "evidence": [
      {
        "link": "https://en.wikipedia.org/wiki/Christopher_Gäng",
        "object": "Mannheim",
        "predicate": "place of birth",
        "probability": 0.8813400910969328,
        "snippet": "Christopher Gäng (born 10 May 1988 in Mannheim)
                    is a German football goalkeeper who currently
                    plays for SG Sonnenhof Großaspach.",
        "source": "Wikipedia",
        "subject": "Christopher Gäng"
      }
    ],
    "object": {
      "link": "https://www.wikidata.org/wiki/Q2119",
      "name": "Mannheim"
    },
    "predicate": {
      "link": "https://www.wikidata.org/wiki/P19",
      "name": "place of birth"
    },
    "probablity": 0.8813400910969328,
    "sentences": [
      "Christopher Gäng (born 10 May 1988 in Mannheim) is a German
      football goalkeeper who currently plays for SG Sonnenhof
      Großaspach."
    ],
    "subject": {
      "link": "https://www.wikidata.org/wiki/Q474877",
      "name": "Christopher Gäng"
    },
    "type": "verified"
  }
]
```

Listing 5: Example response from Prometheus Fact Checker.

EXAMENSARBETE Constructing a Multilingual Relation Extraction System Using Neural Networks
STUDENT Erik Gärtner & Axel Larsson
HANDLEDARE Pierre Nugues (LTH) & Håkan Jonsson (Sony Mobile)
EXAMINATOR Jacek Malec (LTH)

Teknik bekämpar falska nyheter

POPULÄRVETENSKAPLIG SAMMANFATTNING **Erik Gärtner & Axel Larsson**

Falska nyheter är något som cirkulerar på Internet i allt högre grad och orsakar stor skada mot samhället. Vi har byggt ett system som kan användas för att motverka denna trend genom att automatiskt extrahera påståenden från texter som sedan kan kontrolleras mot känd fakta.

Faktakontroll av nyheter är i sig inget nytt men idag konsumerar allt fler människor sina nyheter via sociala medier och undkommer härvid källkritik och faktakontroll som utförs av journalister. Den stora mängden nyheter gör det idag omöjligt att manuellt kontrollera allt. Självklart vore det bra om dessa nyheter automatiskt kunde kontrolleras med datorer men dessa har svårt att förstå vanliga texter. De behöver mer strukturerad data.

Vi har skapat ett system som kan extrahera och strukturera påståenden från vanlig text och som dessutom gör det mycket snabbt. Att gå igenom hela engelska Wikipedia, med mer än 5 miljoner artiklar tar bara 20 minuter med vårt system. Systemet kör på ett tiotal datorer samtidigt för att kunna uppnå denna prestanda.

Det bästa med systemet är att det är självlärande. Genom *maskininlärning* och *språkteknologi* kan vi lära systemet att känna igen olika typer av påståenden. Vi visar systemet exempel på hur påståenden brukar se ut och utifrån det lär sig systemet att känna igen nya påståenden. Internt använder systemet ett *neuralt nätverk* för att skilja mellan olika typer av påståenden, t.ex. om ett påstående beskriver var en person är född eller vem denna är gift med.

Vi lärde systemet genom att låta det bearbeta Wikipedia, vilket i detta fall är en lämplig källa då den är gratis och innehåller flera miljoner artiklar inom alla ämnensområden. Till sin hjälp använde systemet Wikidata som är en öppen databas av fakta – fakta som gav systemet ledtrådar till vilka fakta som systemet kunde förvänta sig att hitta i Wikipedia.

Våra resultat visar att systemet är pålitligt för vissa typer av påståenden. Givet tid är det möjligt att expandera systemet för andra typer av påståenden. Som demonstration konstruerade vi ett plugin till webbläsare som automatiskt kontrollerar påståenden på en hemsida mot tidigare etablerade fakta.

Systemet är dessutom flerspråkigt vilket betyder att fakta och påståenden från olika språk kan kontrolleras och sammanfogas till en enhetlig databas. Detta är viktigt då det i vissa länder finns en stark inhemsk propaganda på det egna språket och inte engelska.

I nuläget är systemet på forskningsstadiet men i framtiden lär denna typ av system komma att användas i många olika sammanhang. Dels för att kontrollera fakta i texter, men även för att bygga databaser till virtuella assistenter och liknande system där informationen främst finns i vanliga texter. Kanske kommer din doktor om 50 år att vara ett system som lärt själv från medicinsk kurslitteratur.